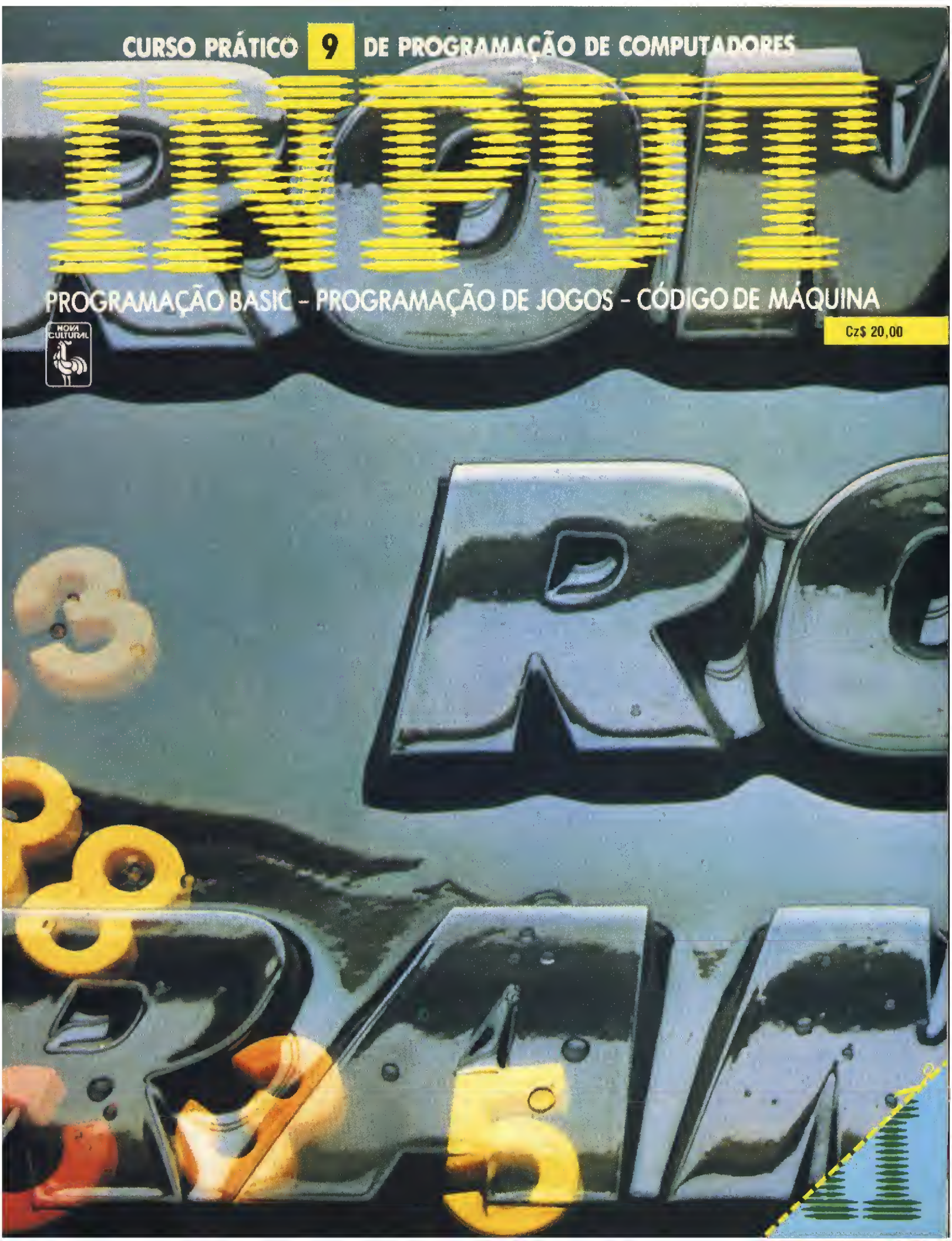
CURSO PRÁTICO 9 DE PROGRAMAÇÃO DE COMPUTADORES

PASCAL

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

NOVA CULTURAL

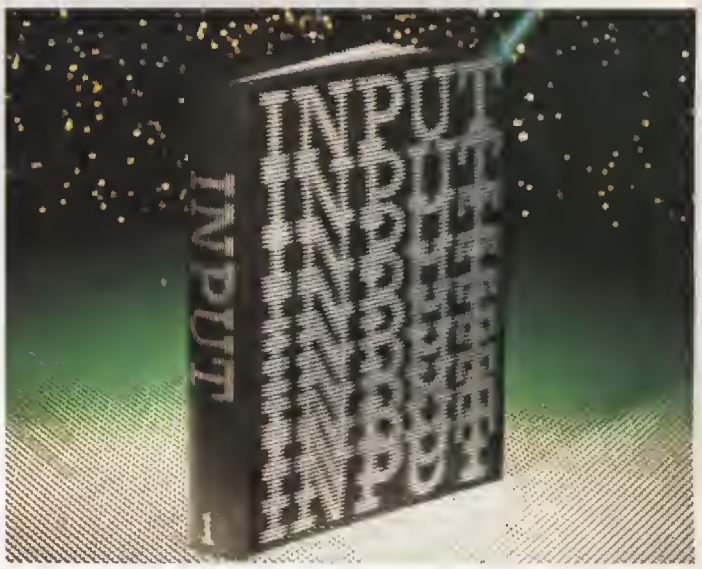
Cz$ 20,00



INPUT

Vol. 1 N.º 9

NESTE NÚMERO


INPUT
INPUT
INPUT

E AGORA... O QUE FAZER?

- O COMANDO INPUT E AS MENSAGENS DE PRONTIDÃO
- MÉTODOS MAIS RÁPIDOS COM INKEY$ OU GET
- PROGRAMAS DE DESENHOS NA TELA
- ENTRADAS DE SENHAS SECRETAS

Grande parte dos programas para micros exige que o usuário digite dados e outras informações necessárias ao seu funcionamento. A maneira como isso é feito tem uma importância fundamental. Veja por quê.

Com muito poucas exceções, os computadores não funcionam inteiramente sozinhos, dando saída a informações. Quase todos os programas, desde jogos até aplicações comerciais e científicas, exigem do usuário algum tipo de entrada. Entretanto, há informações que podem ser fornecidas diretamente ao computador, e existem inúmeras maneiras pelas quais isso pode ser feito, para os vários tipos de programa.

A informação pode ser tão simples quanto a pressão sobre uma tecla de controle do cursor para direcionar uma base de mísseis. Mais comumente, porém, consiste na digitação de números ou textos como acontece, por exemplo, em programas para bancos de dados. No caso de um jogo, a maior exigência é que o computador responda de modo rápido.

INPUTS SIMPLES

Um dos primeiros programas executados pelos iniciantes e que ilustra o uso do comando **INPUT** é o seguinte:

TTSSMC

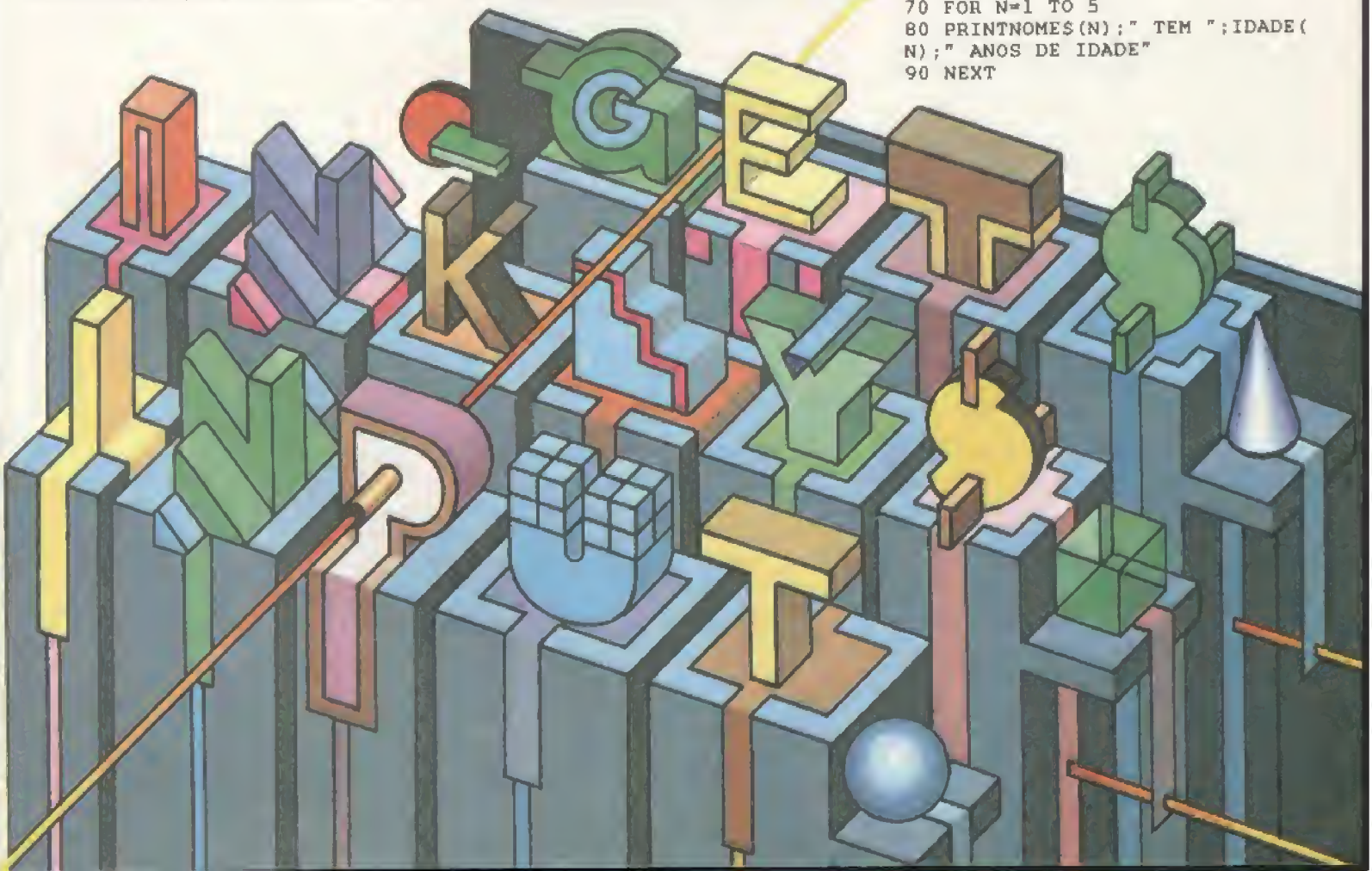
```
10 PRINT"QUAL E O SEU NOME?"
20 INPUT N$
30 PRINT"OLA, ";N$
```

Ao deparar-se com um **INPUT** dentro de um programa, o computador pára o processamento e espera até que o usuário digite algo. Após terem sido pressionados <**ENTER**> ou <**RETURN**>, a mensagem digitada (qualquer que seja ela) é colocada em uma variável — **N$**, no caso do programa acima.

Neste exemplo, a variável é um cordão alfanumérico, mas variáveis numéricas também podem ser utilizadas. No programa seguinte, mostraremos como se faz uma lista de cinco nomes e idades, empregando comandos INPUT simples. Esse programa utiliza dois conjuntos para armazenar informações (mais adiante, abordaremos numa lição específica a questão dos conjuntos e de como funcionam).

TTMSX

```
5 DIM NOMES$(5),IDADE(5)
10 FOR N=1 TO 5
20 PRINT"NOME:"
30 INPUT NOME$(N)
40 PRINT"IDADE:"
50 INPUT IDADE(N)
60 NEXT N
70 FOR N=1 TO 5
80 PRINTNOME$(N);" TEM ";IDADE(
N);" ANOS DE IDADE"
90 NEXT
```



```
 5  DIM n$(5,10): DIM a(5)
10  FOR n=1 TO 5
20 PRINT "Nome: "
30 INPUT n$(n)
40 PRINT "Idade: "
50 INPUT a(n)
55 CLS : NEXT n
60 FOR n=1 TO 5
70 PRINT n$(n);" tem ";a(n);"
 anos de idade"
80 NEXT n
```

Nos computadores da linha ZX-81, digite tudo com letras maiúsculas. Omita as linhas 5 e 55, e acrescente:

```
5 DIM N$(5,10)
7 DIM A(5)
55 CLS
57 NEXT N
```

A informação fornecida ao usuário deve estar correta. Da mesma forma, o usuário precisa entrar exatamente o tipo de informação solicitada. Se você digitar um nome quando for perguntada uma idade, o programa será interrompido. No entanto, rodar o programa mais uma vez leva quase sempre à perda de tudo o que já foi entrado, obrigando o operador a um duplo trabalho. Isso não constitui problema se forem apenas cinco entradas; mas quando se trata de um programa extenso, esse esforço pode ser desgastante.

UTILIZE MENSAGENS DE PROMPTIDÃO

Embora pareça simples, o programa acima tem uma característica importante, que é a *mensagem de prontidão* contida nas duas declarações **PRINT**. Essa mensagem faz com que o programa diga ao usuário que tipo de informação ele deve digitar pelo teclado.

Se as linhas 20 e 40 do programa fossem apagadas, ainda assim ele funcionaria, pois, na maioria dos computadores, toda ocorrência de um **INPUT** provoca o aparecimento de algum sinal de prontidão na tela (por exemplo, um ponto de interrogação).

Se você já está familiarizado com computadores, o sinal de prontidão do **INPUT** é suficiente, quase sempre, para lembrá-lo de que alguma coisa precisa ser entrada. Ele é insuficiente, porém, para lhe dizer qual o tipo de informação a ser digitada. Imagine, por exemplo, que o programa lhe é estranho, que talvez você não tenha experiência em lidar com o computador ou que não tenha conhecimento de como ele funciona. Neste caso, um ponto de interrogação causa apenas estupefação...

A mensagem de prontidão é necessária até mesmo nos programas mais simples, pois o usuário tende a esquecer facilmente a forma exata pela qual a informação deve ser entrada. Por exemplo, a entrada de datas. A maior parte dos programas de contabilidade, além de muitos outros, requer a entrada de datas pelo usuário. Em alguns casos, essa informação é utilizada pelo programa, seja em rotinas de pesquisa, quando se deseja encontrar um dado específico, seja na ordenação cronológica de uma série de eventos, por exemplo. Neste caso, é preciso estabelecer que uma data seja sempre entrada segundo uma forma padronizada.

Um método comum é utilizar seis dígitos para uma data, ou seja, dois dígitos para o dia, dois para o mês, e dois para o ano. Muitas vezes empregam-se barras ou pontos para separar os três períodos. Uma rotina típica de entrada de dados através de um **INPUT** seria:

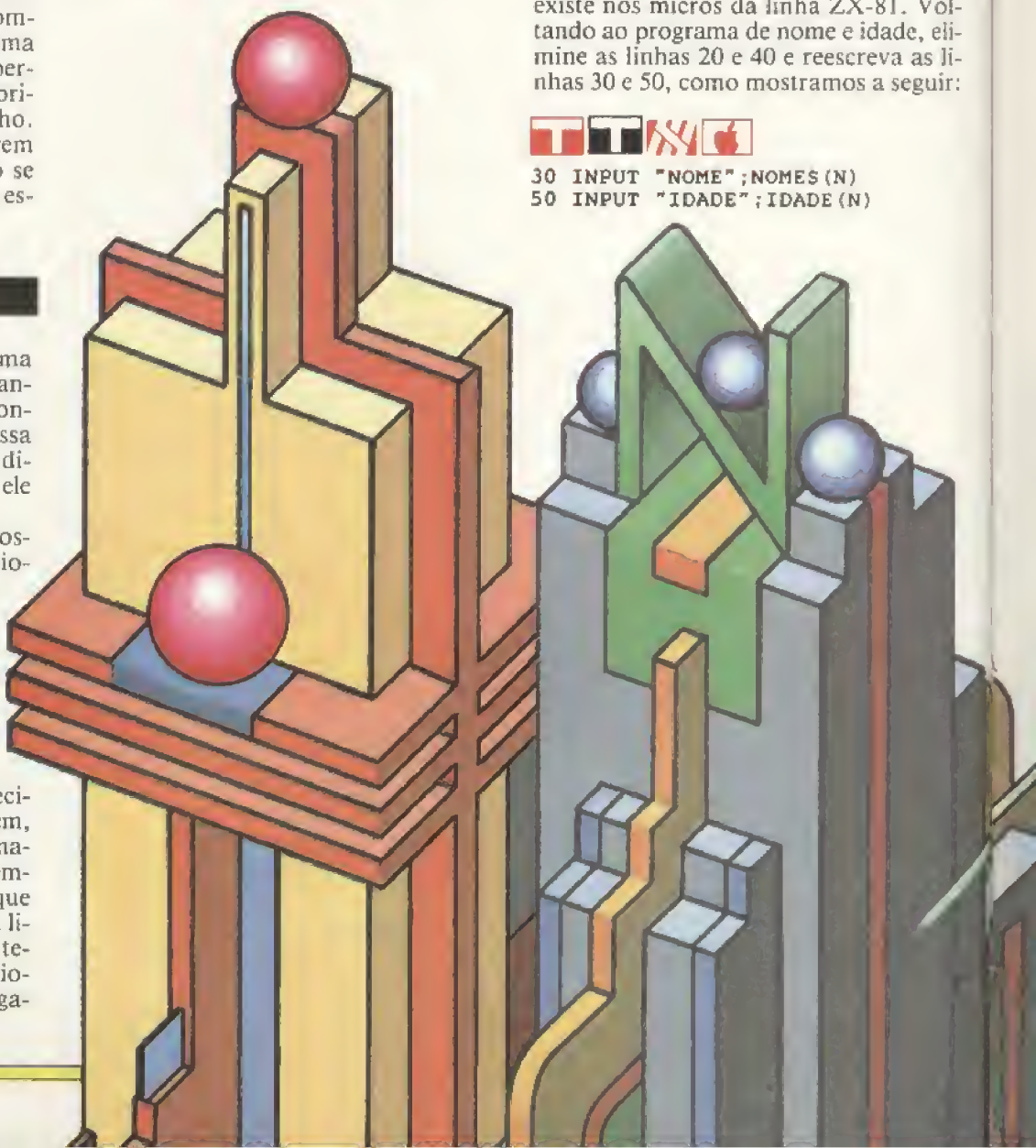
```
100 PRINT "DIGITE DATA (EM FORM
ATO DD/MM/AA)"
110 INPUT D$
```

A mensagem serve não somente para lembrar o usuário de que uma data é necessária, como também para definir a forma como ela deve ser entrada (o que se denomina *formato de entrada* do dado). Se você quiser entrar, por exemplo, 3 de setembro de 1945, deverá digitar: 03/09/45.

O formato de entrada poderá ser qualquer um que o programa desejar, desde que o usuário seja informado, por meio de uma mensagem adequada.

A utilização do comando **INPUT**, como foi mostrada até agora, é bastante simples. Mas ela pode ser ainda mais simplificada no caso de alguns computadores, que permitem a inclusão da mensagem de prontidão no próprio comando **INPUT**. Esse recurso só não existe nos micros da linha ZX-81. Voltando ao programa de nome e idade, elimine as linhas 20 e 40 e reescreva as linhas 30 e 50, como mostramos a seguir:

```
30 INPUT "NOME";NOME$(N)
50 INPUT "IDADE";IDADE(N)
```



Na maioria dos micros (com exceção do TRS-Color), se não for colocado um espaço em branco antes do segundo sinal de aspas, na mensagem de prontidão, o ponto de interrogação aparecerá colado a ela. Por isso, se você quiser dar maior clareza à mensagem, coloque o espaço em branco adicional.

Com exceção do Sinclair Spectrum, é possível compactar ainda mais a rotina de entrada de dados. Assim, as linhas 30 e 50 do programa anterior poderiam ser substituídas por apenas uma:

```
30 INPUT"DIGITE NOME E IDADE ";
NOMES(N),IDADE(N)
```

(Não se esqueça de apagar a linha 50.)

No Spectrum, é necessário colocar somente uma variável por comando **INPUT**, de modo que o usuário deve pressionar duas vezes o <**ENTER**>. No caso do TRS-Color, do TRS-80, do MSX e do Apple II, além de se poder utilizar o método do Spectrum, a linha única oferece a possibilidade de se digitar nome e idade separados apenas por uma vírgula, e de se pressionar o <**ENTER**> somente no final. Pode-se ainda colocar qualquer número de variáveis — separadas por vírgulas em uma única linha de **INPUT**.

Ao se utilizar apenas uma linha de **INPUT** para várias entradas, não se deve esquecer de incluir instruções sobre todas as informações pedidas, bem como os seus formatos. Além disso, é necessário lembrar o usuário de que os dados devem ser separados uns dos outros por vírgulas, e que só no final se deve pressionar a tecla <**ENTER**>. Se essa norma não é seguida a maioria dos micros interrompe o programa, colocando um duplo sinal de interrogação na linha seguinte.

No Sinclair é possível dividir a mensagem de prontidão em mais de uma, na mesma linha. Se cada mensagem de prontidão for mantida entre aspas, o computador se limitará a imprimi-las, sem tratá-las como variáveis.

Entretanto, note que não é possível fazer isso nos computadores das linhas TRS-Color, Apple, TRS-80, MSX e ZX-81. Para melhorar a disposição das informações na tela, as mensagens de prontidão dos **INPUT** podem ser posicionadas por intermédio de comandos **TAB** normais (conforme foi explicado anteriormente, na lição sobre o uso dessa função).

COMO ENTRAR UMA LINHA COMPLETA

O principal problema na utilização do comando **INPUT** surge quando se tenta entrar cordões alfanuméricos que contenham vírgulas e dois pontos, ou espaços em branco no início. Um endereço, por exemplo, normalmente contém vírgulas, de modo que fica impossível usar o **INPUT** para colocá-lo em uma variável alfanumérica simples. Do mesmo modo, a inclusão de espaços no início de uma entrada pode ser útil para a beleza da apresentação na tela.

A dificuldade está em que muitos computadores ignoram espaços à esquerda em um comando **INPUT**, embora não criem problemas com os intervalos entre as palavras. E, quase sempre, eles também ignoram qualquer coisa que ocorra após vírgula, dois pontos ou ponto e vírgula, truncando a informação restante, mesmo que o usuário a tenha digitado. Felizmente, alguns micros possuem um comando que contorna essa dificuldade. Os computadores da linha Sinclair, por sua vez, têm um modo especial de evitá-la.

A solução mais comum é incluir a entrada entre aspas. O ZX-81 e o Spectrum, por exemplo, colocam aspas automaticamente na tela quando uma variável alfanumérica aparece em um comando **INPUT**. Em relação aos outros computadores, o operador deve digitar as aspas como parte da entrada (essa exigência, portanto, deve ser apontada ao usuário, através da mensagem de prontidão).

Uma outra solução, disponível nos computadores TRS-80, TRS-Color e MSX, consiste em utilizar o comando **LINE INPUT**. Este é empregado exatamente do mesmo modo que o **INPUT**.

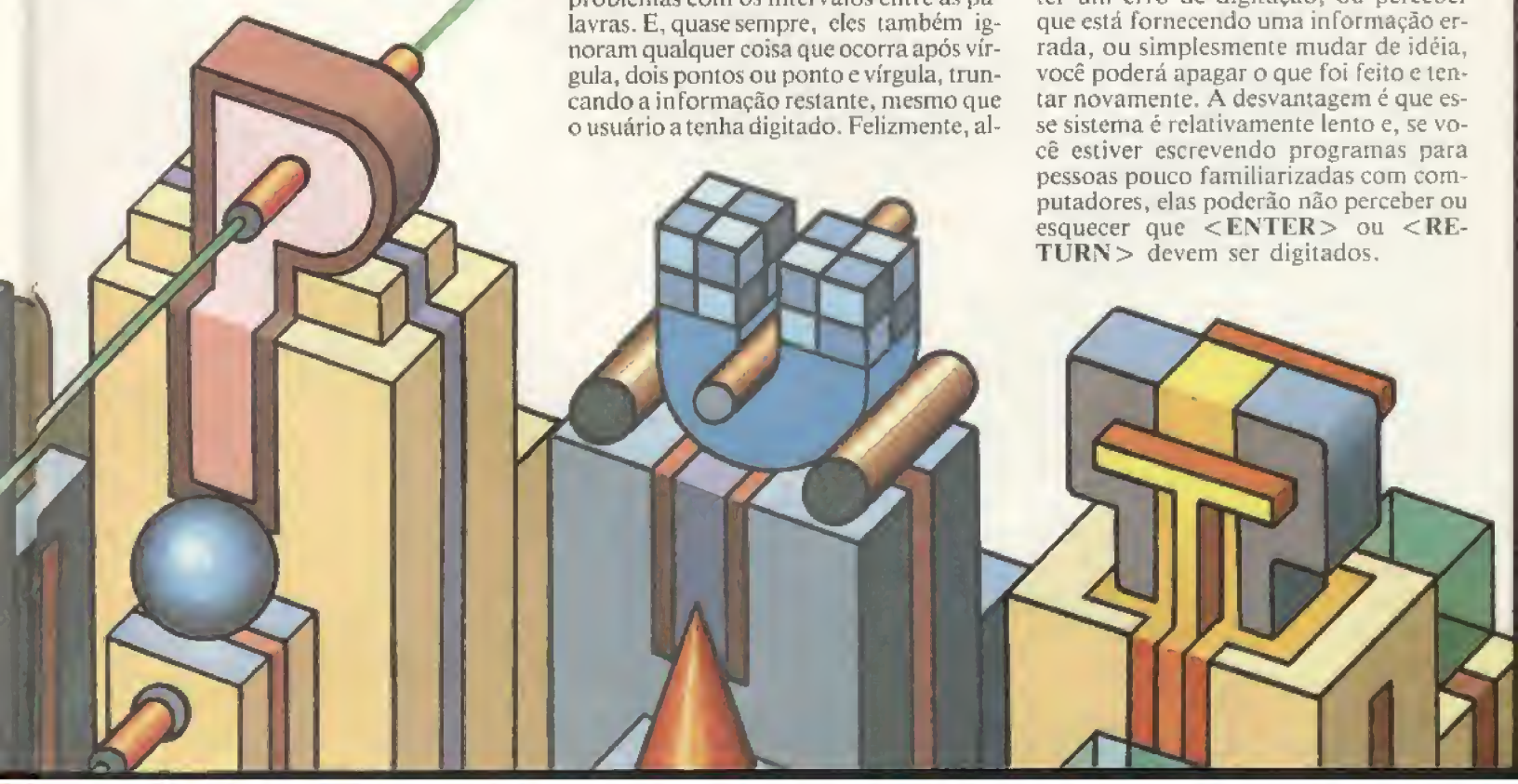
TRS

```
10 LINE INPUT "POR FAVOR, DIGIT
E O SEU ENDERECO ";AS
```

A vantagem é que se pode colocar, dentro da variável, qualquer coisa até o <**RETURN**>, incluindo vírgulas e espaços. Isto é mais garantido, pois você não precisará lembrar o usuário a digitar entre aspas.

ACELERE AS ENTRADAS

Uma das qualidades do comando **LINE INPUT** consiste em permitir que você altere o que estiver entrando, até o momento de pressionar <**RETURN**> ou <**ENTER**>. Assim, se você cometer um erro de digitação, ou perceber que está fornecendo uma informação errada, ou simplesmente mudar de idéia, você poderá apagar o que foi feito e tentar novamente. A desvantagem é que esse sistema é relativamente lento e, se você estiver escrevendo programas para pessoas pouco familiarizadas com computadores, elas poderão não perceber ou esquecer que <**ENTER**> ou <**RETURN**> devem ser digitados.



Em programas que utilizam uma porção de menus ou respostas do tipo "sim ou não", a necessidade de pressionar uma tecla extra diminui a velocidade de execução, além de praticamente duplicar o número de teclas a serem pressionadas. Para evitar que isto aconteça, existe uma maneira de levar o computador a detectar automaticamente a tecla que está sendo pressionada, sem que seja necessário acionar depois as teclas <ENTER> ou <RETURN>

O comando utilizado é o **INKEY$** nos computadores Sinclair, TRS-80, TRS-Color e MSX, e o **GET** nos da linha Apple II.

O efeito da função **INKEY$**, que já estudamos anteriormente, é provocar uma varredura no teclado para averiguar se alguma tecla está sendo pressionada nesse instante. Se este for o caso, a função trará o caractere correspondente de volta ao programa principal. No caso de nenhuma tecla estar sendo pressionada no momento em que o **INKEY$** é encontrado, um cordão vazio é trazido de volta. Assim o **INKEY$** é usado dentro de um comando **IF**, que testa continuamente se o valor retornado é diferente do vazio:

TRS-80 / MSX

```
100 LET A$=INKEY$:IF A$="" THEN
 GOTO 100
```

Já nos micros da linha Apple II, o **GET** faz o computador esperar até que a tecla seja pressionada, não sendo necessário colocá-lo dentro de um laço de espera, como nos outros casos:

```
100 GET A$
```

Qualquer variável alfanumérica pode ser usada — **A$** é apenas um exemplo. A máquina pára na linha 100. Se você teclar o R, o **A$** conterá a letra R. Você pode entrar um número, ou um espaço, ou qualquer outro caractere, até mesmo teclas de controle como <ENTER>. Embora seja virtualmente qualquer coisa, a entrada pode ter apenas um caractere. Assim que a tecla for pressionada, o programa prosseguirá.

DESENHE NA TELA

Agora examine o seguinte programa, que utiliza quatro teclas para desenhar em alta resolução na tela (esse programa não funciona no ZX-81 e no TRS-80, que não dispõem de alta resolução gráfica). Para desenhar no Apple II e no TRS-Color são utilizadas as teclas, Z, X, P e L; no MSX, as teclas a empregar são as de controle do cursor. Você também pode usar a tecla 2 para pedir um desenho na cor de fundo da tela (criando, portanto, uma linha invisível), ou a tecla 1, para retornar à linha do desenho visível.

Uma linha pode ser movimentada em diversas direções. Para isso, basta acionar algumas teclas indicadas pelo programa.

Com um programa simples, que controla apenas algumas teclas, é possível desenhar imagens como as da ilustração.

TRS-80

```
10 PMODE 4,1:PCLS:SCREEN 1,1
20 LET X=100:LET Y=100
40 LET X1=X:LET Y1=Y
50 LET A$=INKEY$
60 IF A$="P" THEN LET Y=Y1-4
70 IF A$="L" THEN LET Y=Y1+4
80 IF A$="Z" THEN LET X=X1-4
90 IF A$="X" THEN LET X=X1+4
100 IF A$="1" THEN COLOR 5
110 IF A$="2" THEN COLOR 0
120 IF A$=" " THEN STOP
130 LINE(X,Y)-(X1,Y1),PSET
140 GOTO 40
```

Sinclair

```
 10 INK 2
 20 PLOT 127,87
 30 IF INKEY$="p" THEN  DRAW 0
 ,2
 40 IF INKEY$="l" THEN  DRAW 0
 ,-2
 50 IF INKEY$="z" THEN  DRAW -
 2,0
 60 IF INKEY$="x" THEN  DRAW 2
 ,0
 70 IF INKEY$="1" THEN  INK 2
 80 IF INKEY$="2" THEN  INK 7
 90 IF INKEY$=" " THEN  STOP
100 PAUSE 10
110 GOTO 30
```

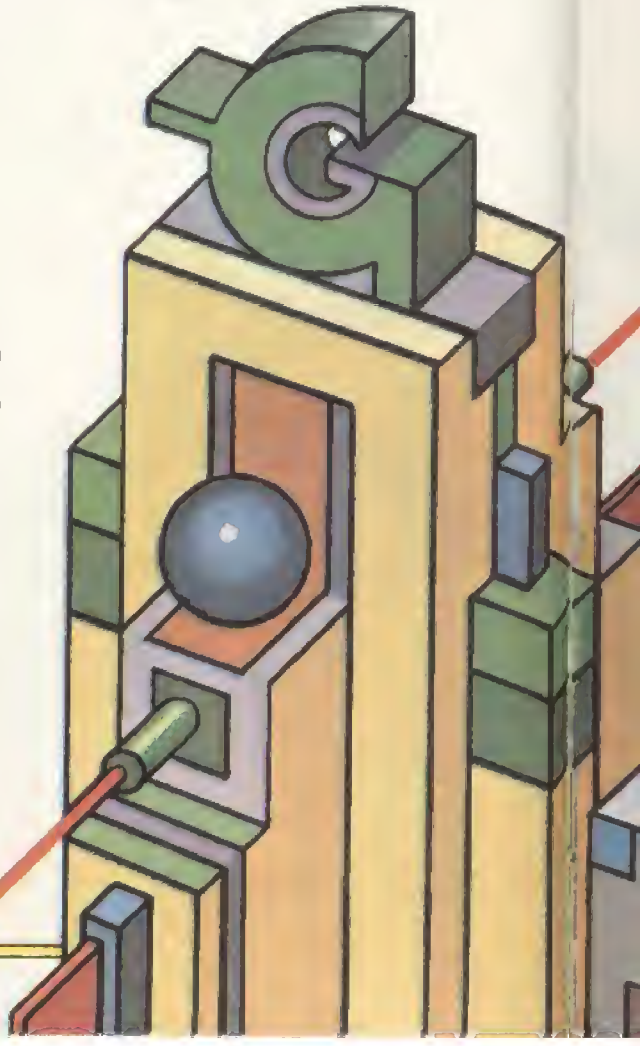
MSX

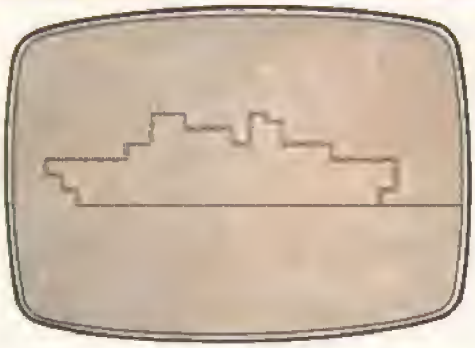
```
10 SCREEN2
20 LETX=100:LETY=100
30 LETX1=X:LETY1=Y
40 A$=INKEY$
50 IFA$=CHR$(30)THENY=Y1-4
60 IFA$=CHR$(31)THENY=Y1+4
70 IFA$=CHR$(29)THENX=X1-4
80 IFA$=CHR$(28)THENX=X1+4
90 IFA$="1"THENCOLOR 4,4
100 IFA$="2"THENCOLOR 15,4
110 IFA$=CHR$(32)THENSTOP
120 LINE(X,Y)-(X1,Y1)
130 GOTO30
```

Apple II

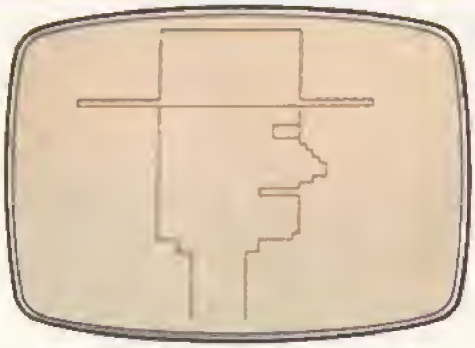
```
10  ONERR  GOTO 150
20  HGR : HCOLOR= 7
30 X = 100:Y = 100
40 X1 = X:Y1 = Y
50  GET A$
60  IF A$ = "P" THEN Y = Y1 - 2

70  IF A$ = "L" THEN Y = Y1 + 2
```





Um barco, um míssil, uma garrafa: se você quiser executar desenhos como esses, siga as instruções desenvolvidas no texto.



Trace as diagonais como se fossem uma escada de minúsculos degraus. Não pressione duas teclas ao mesmo tempo.

```
80  IF A$ = "Z" THEN X = X1 - 2

90  IF A$ = "X" THEN X = X1 + 2

100  IF A$ = "1" THEN  HCOLOR=
7
110  IF A$ = "2" THEN  HCOLOR=
0
120  IF A$ = " " THEN  END
130  HPLOT X,Y TO X1,Y1
140  GOTO 40
150 X = X1:Y = Y1: PRINT  CHR$
(7);: RESUME
```

Esse tipo de rotina é muito útil para gráficos e jogos. A linha é traçada enquanto as teclas são pressionadas. Mas observe que o computador aceita uma tecla de cada vez; já as linhas diagonais são de traçado mais difícil, pois consistem em uma série de pequenos degraus.

Os comandos **INKEY$** ou **GET** são também muito úteis em qualquer programa que empregue menus. O programa seguinte imprime um menu como parte de um programa de arquivo de dados.

```
5 CLS
10 DATA CRIAR UM ARQUIVO,ENTRAR
 DADOS, VER DADOS,EDITAR,PROCUR
AR,IMPRIMIR,CARREGAR,GRAVAR,FIM
15 RESTORE
20 FORN=1TO9
30 READ OPCOES$
40 PRINTTAB(5);N;TAB(10);OPCOES
$
50 NEXTN
60 PRINT:PRINTTAB(5)"SUA OPÇÃO
=>"
70 A$=INKEY$:IFA$=""THEN70
80 IFA$="1"THENGOSUB1000
90 IFA$="2"THENGOSUB2000
100 IFA$="3"THENGOSUB3000
110 IFA$="4"THENGOSUB4000
120 IFA$="5"THENGOSUB5000
130 IFA$="6"THENGOSUB6000
140 IFA$="7"THENGOSUB7000
150 IFA$="8"THENGOSUB8000
160 IFA$="9"THENGOSUB9000
170 GOTO5
```

Altere a linha 70 do programa anterior para:

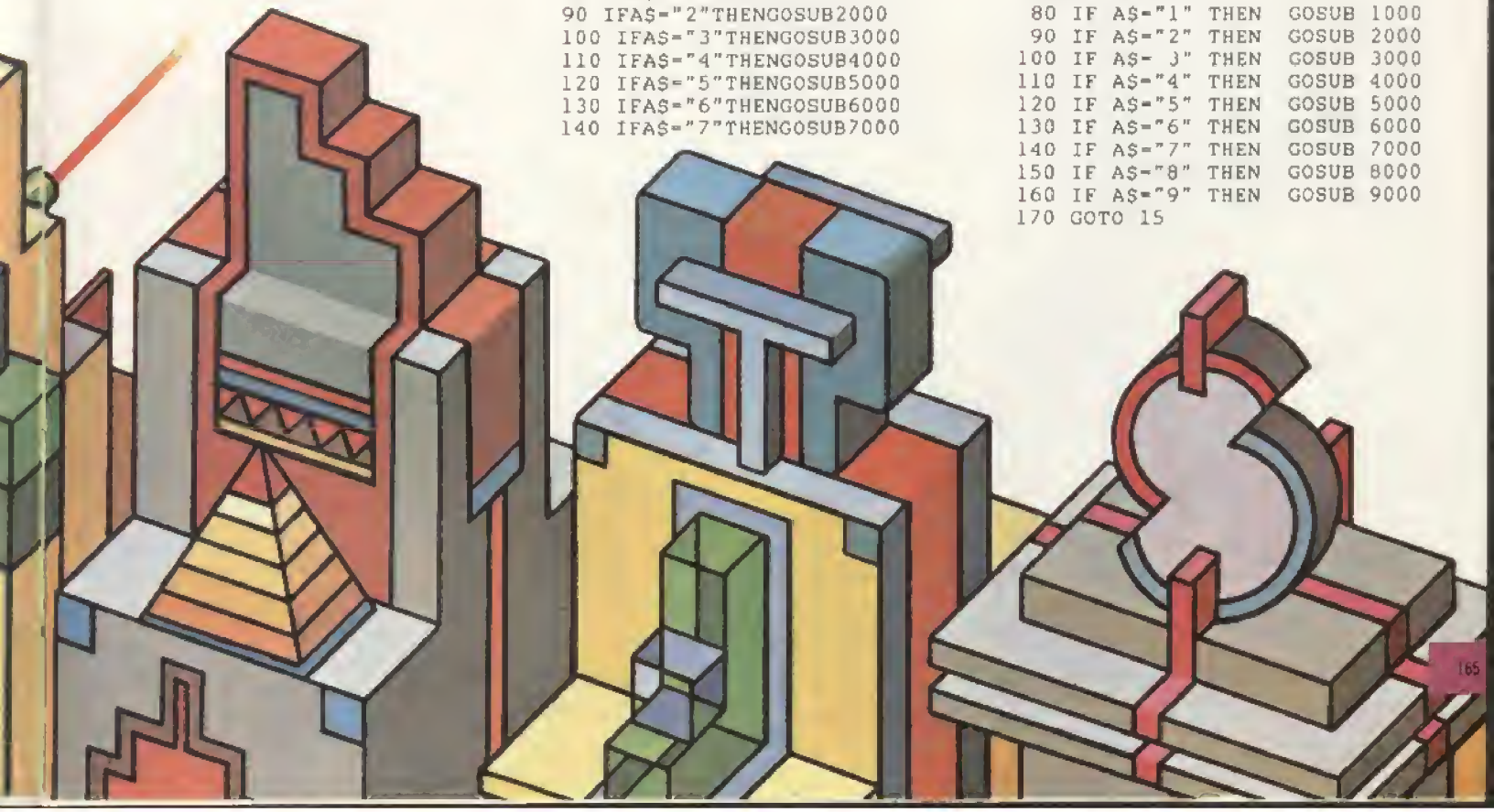
```
70 LET A$=INKEY$:IF A$="" THEN
GOTO 70
```

Modifique as linhas do programa anterior para:

```
5 HOME
70 GET A$:IF A$="" THEN 70
```

Apague as linhas 10 e 70 do programa anterior, substituindo-as por:

```
 10 DATA "Criar novo arquivo",
 "Entrar com registros","Ver r
 egistros","Editar registros",
 "Procurar registros","Imprimi
 r arquivo","Carregar arquivo"
 ,"Gravar arquivo","Saida"
 15 RESTORE
 20 FOR n=1 TO 9
 30 READ h$
 40 PRINT TAB 5;n;TAB 10;h$
 50 NEXT n
 60 PRINT : PRINT TAB 5;"Sua e
 scolha->"
 70 LET A$=INKEY$: IF A$=""
 THEN  GOTO 70
 80 IF A$="1" THEN  GOSUB 1000
 90 IF A$="2" THEN  GOSUB 2000
100 IF A$= 3" THEN  GOSUB 3000
110 IF A$="4" THEN  GOSUB 4000
120 IF A$="5" THEN  GOSUB 5000
130 IF A$="6" THEN  GOSUB 6000
140 IF A$="7" THEN  GOSUB 7000
150 IF A$="8" THEN  GOSUB 8000
160 IF A$="9" THEN  GOSUB 9000
170 GOTO 15
```



Com esse programa, o computador vai direto à sub-rotina pertinente assim que uma tecla é acionada — a menos que você pressione uma outra tecla entre os números de 1 a 9. Neste caso, a linha 170 imprimirá novamente o menu e este se repetirá na tela.

UMA ROTINA PARA SENHAS

O programa anterior funcionará perfeitamente enquanto existirem menos de nove opções. Mas se você tentar teclar 10, o computador entenderá que você está entrando o 1, pois o programa prosseguirá automaticamente assim que tiver sido teclado o primeiro dígito. Mas existe uma maneira de entrar palavras inteiras. E como não é mostrado nada na tela, ela é ideal para se entrar uma senha ou um "código secreto", que pode ser usado para limitar o acesso dos usuários a um determinado programa. Cada dígito é entrado utilizando-se o **INKEY$** ou o **GET**, sendo o caractere resultante acrescentado ao último dígito entrado. Eis o programa:

TK

```
10 PRINT "DIGITE A SENHA"
20 LET K$=INKEY$:IF K$="" THEN
GOTO 20
30 LET P$=P$+K$
40 IF LEN(P$)<>7 THEN GOTO 20
50 IF P$<>"BANANAS" THEN STOP
60 PRINT "O.K."
70 REM (A SEGUIR VEM O PROGRAMA
)
```

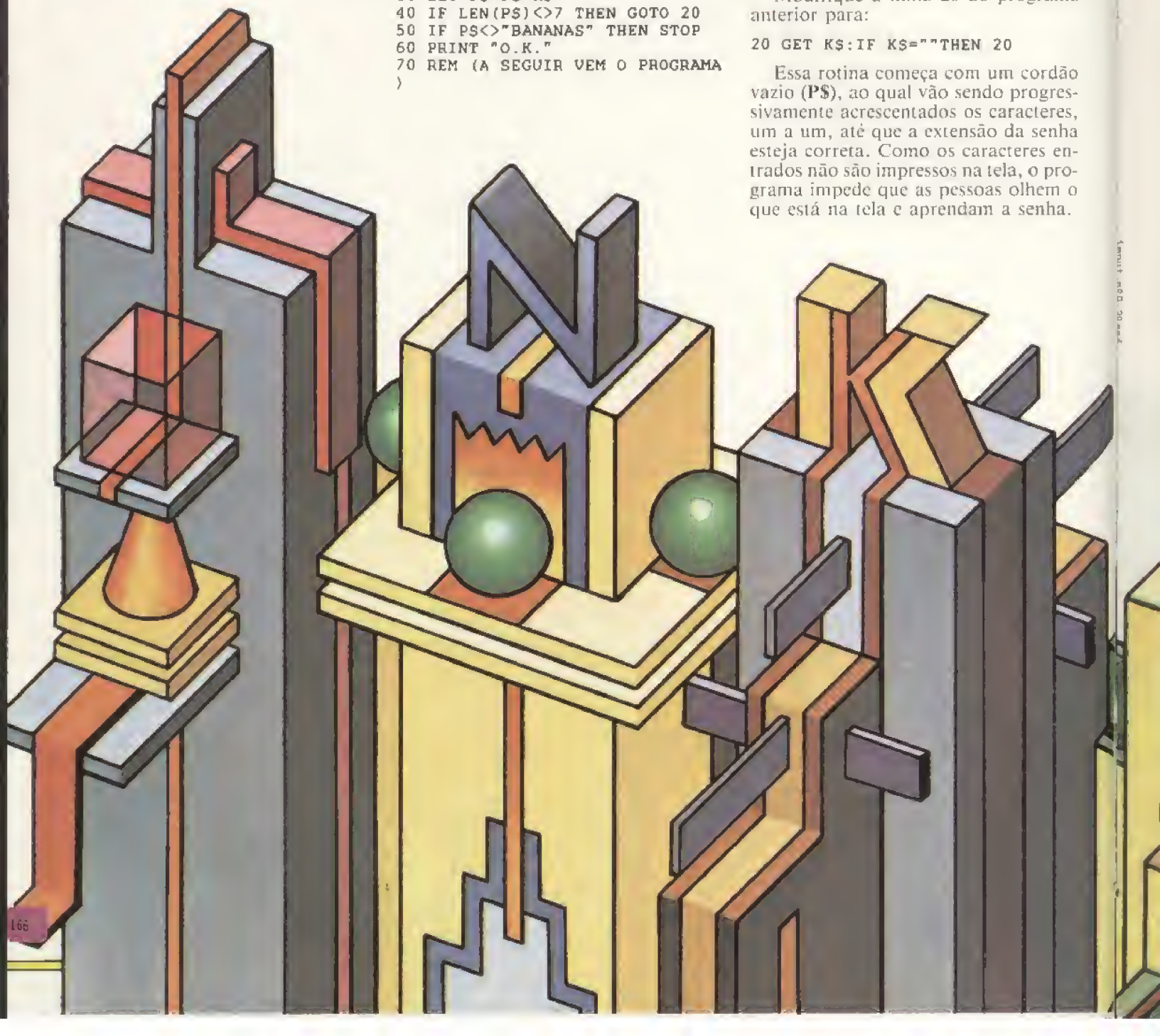
SS

```
10 LET p$=""
20 PRINT "DIGITE A SENHA"
30 PAUSE 0
40 LET k$=INKEY$: IF k$=""
THEN  GOTO 40
50 LET p$=p$+k$
60 IF LEN p$<>7 THEN  GOTO 30
70 IF p$<>"bananas" THEN
STOP
80 PRINT "O.K."
90 REM (Em seguida vem o rest
o do programa"
```

Modifique a linha 20 do programa anterior para:

```
20 GET K$:IF K$=""THEN 20
```

Essa rotina começa com um cordão vazio (**P$**), ao qual vão sendo progressivamente acrescentados os caracteres, um a um, até que a extensão da senha esteja correta. Como os caracteres entrados não são impressos na tela, o programa impede que as pessoas olhem o que está na tela e aprendam a senha.



O **INKEY$** (e outros comandos similares) é igualmente empregado para "congelar" um programa. Isto é útil quando uma tela repleta de informações deve ser examinada. Após a parte do programa que imprime a informação, coloca-se um comando **INKEY$** ou **GET** no programa de modo a bloquear a listagem antes de limpar a tela.

COMPUTADORES QUE NÃO TÊM O INKEY$

Infelizmente, o interpretador Applesoft BASIC, dos micros da linha Apple II, não dispõe de uma função tão poderosa como o **INKEY$**, para efetuar "varreduras" no teclado. Existe apenas o comando **GET$**, mas este bloqueia o andamento do programa toda vez que é encontrado por ele. Dessa forma, se quisermos, por exemplo, fazer programas de jogos em que um missil ou um disco voador precisa continuar se movendo pela tela enquanto o jogador não pressiona a tecla que comanda um disparo, é necessário "imitar" a ação do **INKEY$**, executando o pequeno programa a seguir (não tente entendê-lo ainda; estudaremos os comandos **PEEK** e **POKE** numa lição posterior):

```
100 LET K$=""
110 LET K=PEEK(-16384)
120 IF K<128 THEN RETURN
130 LET K$=CHR$(K-128)
140 POKE -16368,0
150 RETURN
```

Esta sub-rotina, quando chamada (por meio de um **GOSUB 100**), retornará, através da variável **K$**, o caractere pressionado no teclado. Se **K$** retornar com valor nulo, isso significa que nenhuma tecla foi pressionada. Por isso, podemos colocar a chamada à sub-rotina em um laço, assim:

```
30 GOSUB 100
35 IF K$="" THEN GOTO 30
```

O comando **PEEK** examina a locação de memória -16384, que guarda um código numérico. Se esse número for igual a 128 ou maior, significa que uma tecla foi pressionada e que o seu código ASCII será deduzido pela expressão da linha 130 e convertido para o caractere armazenado em **K$**. O comando **POKE** zera uma outra locação da memória, para permitir uma nova varredura.

O interpretador BASIC do TK-2000 não dispõe de uma função tão poderosa como o **INKEY$** para efetuar "varreduras" no teclado. Existe apenas o comando GET (de funcionamento idêntico ao do Apple II), que tem a desvantagem de bloquear o andamento do programa, sempre que é encontrado por ele. Assim, para programarmos jogos em que um míssil ou um disco voador, por exemplo, precisa continuar se movendo pela tela enquanto o jogador não pressiona a tecla que comanda um disparo, é necessário "imitar" a ação do **INKEY$** através de uma rotina diferente da que foi apresentada para os micros da linha Apple II.

A locação 39 da memória RAM do TK-2000 retém um valor numérico correspondente à última tecla pressionada. Assim, a sub-rotina abaixo retorna esse código toda vez que uma tecla for pressionada:

```
100 LET K=0
110 LET K=PEEK(39)
120 IF K=48 THEN GOTO 100
130 RETURN
```

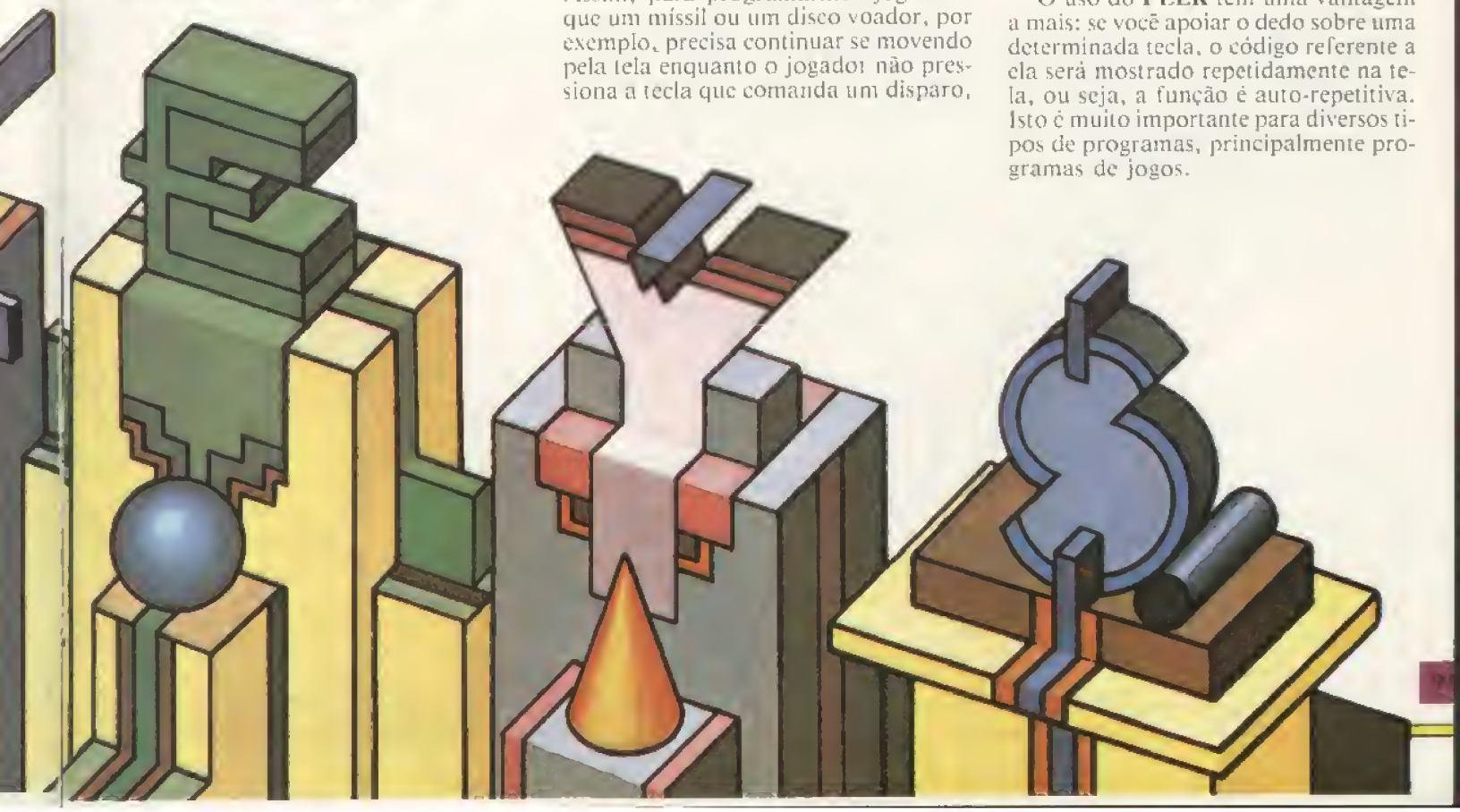
A locação de memória 39, no TK-2000, contém sempre o número 48, enquanto nenhuma tecla for pressionada.

Assim, é fácil incluir uma chamada a essa rotina (**GOSUB 100**) no ponto do programa em que se torna necessário verificar se alguma tecla foi pressionada e, em caso positivo, qual foi essa tecla.

Para descobrir o código gerado por cada tecla do TK-2000, rode o pequeno programa abaixo e pressione sucessivamente as teclas que quer descobrir. O código correspondente será mostrado na tela. Pressione <**CTRL**><**C**> para interromper o programa:

```
10 HOME
20 LET K=PEEK(39)
30 IF K<>48 THEN PRINT K
40 GOTO 20
```

O uso do **PEEK** tem uma vantagem a mais: se você apoiar o dedo sobre uma determinada tecla, o código referente a ela será mostrado repetidamente na tela, ou seja, a função é auto-repetitiva. Isto é muito importante para diversos tipos de programas, principalmente programas de jogos.



QUEBRE A BARREIRA DO SOM

Bipes, explosões, ruídos extraterrenos: seja qual for a sua escolha, todos esses sons podem ser produzidos por programas simples em BASIC, tornando seus jogos mais eletrizantes.

Os programas de jogos de ação incluem, normalmente, diversos tipos de efeitos sonoros — explosões, tiros, zumbidos, pequenas melodias e outros ruídos — que os tornam mais excitantes e atraentes.

Esta lição tem por finalidade proporcionar a você uma pequena "biblioteca" de sons adequados a programas de jogos. Você pode utilizá-los tal como estão ou desenvolvê-los livremente, como parte de conjuntos sonoros mais complexos.

Lembre-se de que não existem regras sumárias ou definitivas para se produzir efeitos sonoros. Se o seu jogo precisa, por exemplo, de um ruído semelhante ao de alguém mergulhando numa piscina, você terá que sentar-se em frente ao computador e experimentar várias vezes, até conseguir um efeito convincente. Por outro lado, alguns sons aparentemente improváveis poderão ser bem aproveitados, caso você consiga inventar os tipos adequados de gráficos.

A incorporação de efeitos sonoros em seus programas depende da complexidade destes e da freqüência com que tais efeitos serão utilizados. Assim, ruídos simples que entram apenas uma vez em um programa, devem ser colocados após uma declaração **IF... THEN**. Efeitos repetitivos ou mais complexos, por sua vez, precisam de uma sub-rotina.

O grau de sofisticação dos efeitos sonoros depende não só da habilidade do programador, mas também dos recursos de programação de sons de cada computador. Desse modo, o gerador de som dos micros compatíveis com o Sinclair Spectrum (por exemplo, o TK-90X) é limitado a um simples tom puro (chamado de *hipe*, em jargão computacional) obtido por meio do comando **BEEP**, e cuja freqüência e duração podem ser controlados pelo usuário. Um ruído semelhante pode ser produzido, no microcomputador brasileiro TK-2000, mediante o comando **SOUND**. Em contrapartida, os micros das linhas TRS-Color e MSX possuem um gerador de sons bastante sofisticado, capaz de sintetizar uma série imensa de efeitos diferentes. Esse gerador dispõe de vários comandos específicos na linguagem BASIC. O Apple II, o ZX-81 e o TRS-80 só permitem efeitos sonoros satisfatórios através de programas especiais em linguagem de máquina; por isso não serão tratados nesta lição.

O Sinclair Spectrum conta, como já foi assinalado, com um único recurso para a programação de efeitos sonoros em BASIC: o comando **BEEP**. Simples de programar, esse comando pode ser usado para "envenenar" consideravelmente os seus programas. No TK-2000 temos um comando equivalente: o **SOUND**. Para exemplificar, veja a seguir uma rotina que produzirá uma série de sons idênticos:

- INVENTE O SOM CERTO
- COMO INCORPORAR EFEITOS SONOROS AO JOGO DE LABIRINTOS ALEATÓRIOS
- APRENDA A USAR **BEEP**, **PLAY** E **SOUND**
- EXPLOSÕES, TIROS, DESTRUIÇÃO
- DO RUÍDO ÀS NOTAS MUSICAIS
- MAIS EXPLOSÕES

```
8000 FOR n=1 TO 12
8010 BEEP .03,30
8020 NEXT n
```

Como você pode ver na linha 8010 do programa acima, o **BEEP** é seguido por dois números separados por uma vírgula. O primeiro número determina a duração da nota a ser tocada: quanto maior ele for, mais tempo durará a nota. O valor 1 corresponde à duração de um segundo. Números maiores ou menores do que esse (no caso, frações decimais) funcionarão proporcionalmente.

O segundo número estabelece a freqüência da nota, com o número 0 correspondendo ao dó médio no teclado de um piano. Cada número inteiro acima ou abaixo disso representa um semitom mais alto ou mais baixo — ou seja, é correspondente à tecla mais próxima, em um teclado de piano. Para efeitos sonoros em jogos, freqüências muito baixas, da ordem de −32, são utilizadas para imitar explosões e zumbidos.

```
8000 FOR N=1 TO 12
8010 SOUND 90,3
8020 NEXT N
```

O comando **SOUND** no TK-2000 (veja a linha 8010) funciona da mesma maneira, só que agora o primeiro número determina a freqüência do som, e o segundo a sua duração. Não são permitidos números negativos ou fracionários, nem maiores que 255. A correspondência da freqüência com as notas musicais é um pouco mais complicada, e está ilustrada no manual de programação BASIC do TK-2000.

Os comandos **BEEP** e **SOUND** serão explicados com maior riqueza de detalhes numa próxima lição.

Na rotina acima, o laço **FOR... NEXT** executa uma série de doze notas. Já o programa abaixo contém dois laços, com a variável de controle de cada laço estabelecendo a freqüência das notas. Isto resulta em um efeito que poderá lhe ser útil quando, em um de seus jogos, for necessário criar um som para acompanhar a destruição de um extraterrestre.

```
8000 FOR n=4 TO 0 STEP -1
8010 BEEP .01,n
8020 NEXT n
8030 FOR n=1 TO 4
8040 SOUND .01,n
8050 NEXT n
```

```
8000 FOR N=60 TO 0 STEP -10
8010 SOUND N,3
8020 NEXT N
8030 FOR N=0 TO 60 STEP 10
8040 SOUND N,3
8050 NEXT N
```

Eis aqui uma rotina que emprega o laço **FOR...NEXT** de um modo pareci-

do; o ruído criado, porém, será agora um som de "Congratulações". Você poderá utilizá-lo quando o jogador tiver destruído todos os "monstros" ou para celebrar alguma outra vitória.

```
8000 FOR n=10 TO 60 STEP 5
8010 BEEP .01,n
8015 BEEP .01,n-2
8020 NEXT n
```

```
8000 FOR N=70 TO 150 STEP 5
8010 SOUND N,3
8015 SOUND N-2,3
8020 NEXT N
```

Tente fazer experiências mudando os parâmetros do comando **BEEP** (ou **SOUND**) para produzir efeitos sonoros em seus programas de jogos. Procure utilizá-los também dentro de laços e de sub-rotinas.

MICRO DICAS

EFEITOS SONOROS NO TRS-80

O TRS-80 não tem recursos para a produção de efeitos sonoros, mas é possível fazê-lo emitir alguns sons. Para isto, utiliza-se a interface de saída para o gravador cassete, que gera pulsos no espectro audível, sob controle adequado de um software. Basta ligar o fio que vai do computador até o conector MIC ou AUX, em um sistema de amplificação de som.

O software de produção de som funciona da seguinte maneira: o comando **OUT** envia para a porta de saída do gravador (a de número 255) uma seqüência de bytes 0 e 1. O byte 0 causa um pulso negativo na saída, e o byte 1, um pulso positivo. Alternando-se ambos, temos uma onda de som. Para variar a freqüência do som, varia-se a pausa entre o pulso positivo e o negativo. Com essa técnica, é possível produzir sons em BASIC: basta colocar os comandos OUT dentro dos laços de repetição entre meados de um laço de retardo de tempo:

```
10 FOR I=1 TO 1000
20 OUT 255,0
30 FOR J=1 TO 10:NEXT J
40 OUT 255,1
50 NEXT I
```

Sons mais agudos e efeitos como explosões, sirenas, etc., exigem uma rotina em linguagem de máquina, que pode ser chamada a partir de um programa em BASIC pelo comando **USR**.

SONS PARA O LABIRINTO

Imagine o jogo de labirinto aleatório apresentado na última lição com alguns efeitos sonoros. Note que não há uma versão para o TK-2000. Adicione as linhas 365 e 500 ao programa original e modifique a linha 400.

```
365 BEEP .01,10
400 LET vidas=vidas-1: RESTORE
500: FOR f=0 TO 10: READ a,b:
BEEP a,b: NEXT f: IF vidas>0
THEN  GOTO 260
500 DATA .45,0,.3,0,.15,0,.45,
0,.3,3,.15,2,.3,2,.15,0,.3,0,.
15,-1,.45,0
```

Se você rodar o programa agora, descobrirá que acrescentou uma rotina de som simples mas muito eficaz, que executa uma marcha fúnebre sempre que o jogador perder uma "vida". Tente modificar os valores de a e b para criar efeitos diferentes.

Esse exemplo mostra que você deve ser cuidadoso quando utilizar efeitos sonoros em jogos, já que o computador interrompe o que está fazendo durante a execução do **BEEP**, criando assim algumas pausas inoportunas. Mesmo que você inclua apenas efeitos de curta duração, isso poderá tornar o seu jogo muito lento.

O MSX possui um dispositivo sonoro muito sofisticado que emprega três "vozes" altamente controláveis, ou canais sonoros. Esse dispositivo permite uma grande variedade de ruídos, muitos dos quais podem ser incorporados aos seus programas de jogos.

Tais ruídos podem ser criados por meio das "vozes", isoladamente ou em grupos, de modo a formar diversas combinações. Uma explicação mais detalhada sobre programação de efeitos sonoros no MSX será desenvolvida em um artigo posterior de BASIC.

O MSX utiliza um segundo microprocessador interno somente para produzir sons e efeitos sofisticados: algumas linhas de programa são suficientes para isso. Esse microprocessador é composto por 14 registros que funcionam de maneira semelhante às posições de memória do computador. O som é produzido de acordo com o conteúdo de tais registros, ou seja, suas características dependem dos números armazenados nos registros. Os valores adequados são colocados nesses registros por intermédio do comando **SOUND R, N** (**R** é o número do registro, e **N** o número que determinará as características do som).

O MSX conta ainda com uma linguagem "macromusical" disponível mediante o comando **PLAY**. Quando se deseja programar uma melodia (é necessário entender um pouco de música) é mais fácil utilizar esse comando e não se preocupar com tantos registros. O **PLAY** é, contudo, um pouco lento e não produz ruídos, só notas musicais. Para entendê-lo melhor, leia a seção dedicada ao TRS-Color, mais adiante.

JOGO SONORO NUM LABIRINTO

Se você gravou o programa de criação de labirintos aleatórios para o MSX (apresentado na última lição de programação de jogos), certamente achará interessante acrescentar as três linhas que listamos a seguir.

```
1330 IF X<>LX OR Y<>LY THEN LX=
X:LY=Y:PLAY "V15T25505L64DG"
1340 IF F=1 THEN F=0:SC=SC+(TI-
TIME):TI=TI-10:PLAY "V15T10003B
CDEFGA":GOTO 1220
1500 PLAY "V15T25502L2CR64L4CR6
4L12CR64L2CR64L4D#R64L8DR64L4DR
64L8CR64L4CR6401L8BR6402L2C":LI
=LI-1
```

A primeira linha produz som para o movimento do homenzinho; a segunda, para o momento em que ele encontra o tesouro; a terceira toca uma "marcha fúnebre" quando ele perde uma "vida".

Como dissemos, o comando **PLAY** só é adequado para programação de melodias. Quando desejarmos ruídos (sons

não musicais) ou uma velocidade maior, deveremos utilizar o comando **SOUND**.

Muitos valores (até 13) devem ser colocados nos registros adequados, para a emissão de um determinado som. Comece a aprender como fazê-lo, digitando as seguintes linhas:

```
10 SOUND 7,7
20 SOUND 6,20
30 SOUND 8,15
```

Rode o programa e ouvirá um ruído semelhante ao das ondas do mar quebrando na praia (use a imaginação).

Vejamos os registros utilizados. Na linha 10 colocamos o valor 7 no registro 7. Este último determina os canais de som a serem utilizados e se eles produzirão música ou ruído. O que importa no momento é que o valor 7 ativa os três canais para criar ruídos. O valor 56 ativaria os três para sons musicais. Qualquer outro valor entre 1 e 63 utilizaria os canais de uma maneira mista. Na realidade, deve-se converter o valor utilizado para a forma binária de modo a ficarmos sabendo o que cada canal produzirá: música ou ruído.

Na linha 20, o registro 6 determina a freqüência do ruído produzido, podendo conter valores de 0 a 63. Números menores determinam sons mais agudos. Na linha 30, o registro 8 controla o volume do som produzido pelo canal A, podendo conter um número entre 0 e 15. Geralmente, é melhor colocarmos o valor máximo e ajustarmos o volume da TV. Outras vezes, contudo, podemos criar efeitos interessantes variando o volume. Acrescente as seguintes linhas ao programa para obter um som mais próximo do real. Isto é conseguido variando lentamente o volume nas linhas 40 e 70.

```
30 FOR I=10 TO 15 STEP 1E-03
40 SOUND 8,I
50 NEXT
60 FOR I=15 TO 10 STEP -1E-03
70 SOUND 8,I
80 NEXT
90 GOTO 30
```

Modifique o valor do **STEP** nas linhas 30 e 60 de -1E-3 para .5 e você ouvirá o barulho de um trem a vapor em movimento. O programa listado a seguir permite que você experimente e modifique vários efeitos sonoros, variando os valores dos diversos registros.

```
5 CLS:R=RND(-TIME)
10 INPUT "Número do programa (1
-13): ";P
15 INPUT "Envoltória (1-fixa ,
2-programável): ";E
16 EE=14+E
20 IF E=1 THEN 35
25 INPUT "Forma da onda sonora
(8-15): ";W
26 SOUND 13,W
30 INPUT "Período do ciclo (0-6
5384): ";F
31 H%=F/256:L%=F-256*H%
32 SOUND 11,L%:SOUND 12,H%
35 INPUT "Música ou ruído (M ou
 R): ";M$
36 IF M$="M" THEN SOUND 7,56:A=
0:T=255:GOTO 40
38 IF M$="R" THEN SOUND 7,7:A=6
:T=63:GOTO 40
39 GOTO 35
40 SOUND 8,EE
60 ON P GOSUB 80,90,100,110,120
,130,140,150,160,170,180,190,20
0,210
70 SOUND 8,0:END
80 SOUND 1,5:FOR Z=1 TO T STEP
.3
85 SOUND A,Z:NEXT:RETURN
90 SOUND 1,0:FOR Z=T TO 1 STEP
-.3
95 SOUND A,Z:NEXT:RETURN
100 SOUND 1,0:FOR Z=1 TO 20 STE
P .1
105 SOUND A,ABS(SIN(Z)*T/2)
107 NEXT:RETURN
110 SOUND 1,1:FOR Z=1 TO 200
115 SOUND A,RND(1)*(T+1):NEXT
117 RETURN
120 IF A=0 THEN A=1:T=15
122 FOR Z=1 TO 50
125 SOUND A,T AND ABS(TAN(Z)+5)
127 NEXT:RETURN
130 SOUND A,T/2:SOUND 1,2
```

A linha 5 limpa a tela e prepara o gerador de números aleatórios. Os diversos **INPUT** das linhas seguintes pedem informações a respeito do efeito desejado. Nas linhas 80 a 207 estão treze diferentes sub-rotinas que geram treze efeitos diversos.

Essas sub-rotinas cuidam principalmente da freqüência dos sons — isto é, se eles são agudos ou graves — e de como essa freqüência vai variar com o tempo. Caso se trate de som musical, é utilizado o canal A, cuja freqüência é estabelecida pelos registros 0 (ajuste fino) e 1 (ajuste grosseiro). Em caso de ruído, a freqüência é determinada pelo registro 6. A variável A contém o valor adequado do registro de freqüência, conforme a opção seja música ou ruído. A variável T varia também conforme essa opção, compatibilizando os valores máximos que cada registro de freqüência comporta.

Conforme o valor do registro 7 (linhas 36 e 38) teremos ruídos ou sons musicais. O valor 7 determina ruído, e o valor 56, música (nos três canais).

Uma das características do dispositivo sonoro do MSX é a possibilidade de programar o formato da onda sonora

ou "envoltória". Isto significa que o volume do som muda ao longo do tempo de acordo com uma curva cujo formato pode ser escolhido por nós (formato de serra, triângulo, pulso, etc). Normalmente, o volume é fixado de acordo com o valor do registro 8 (para o canal A). Se colocarmos nesse registro o valor 16 (linha 40) teremos à nossa disposição diversos tipos de envoltórias (veja o formato delas em seu manual). O registro 13 seleciona a forma da envoltória, e os registros 11 e 12 determinam o período do ciclo. Este último valor corresponde à velocidade com que o volume vai mudar. Valores baixos correspondem a altas velocidades.

Comece então a experimentar. Escolhendo o programa 10, por exemplo, com envoltória fixa e sons musicais, você ouvirá um canto de pássaro. As variações são quase ilimitadas.

Para entender melhor como todos esses valores modificam as características do som produzido, faça o seguinte: primeiro ouça todos os treze programas utilizando envoltórias fixas e sons musicais. Depois comece a ouvi-los com as diferentes envoltórias programáveis, mantendo o mesmo programa para poder comparar. Inicialmente, varie apenas a forma da onda, mantendo o valor 1000 para o período do ciclo. A seguir, varie o período, conservando o resto constante. Depois de experimentar os sons musicais, repita a mesma seqüência para os ruídos.

Existem dois comandos no BASIC do TRS-Color que podem ser usados para produzir efeitos sonoros: o **SOUND** e o **PLAY**. Ambos utilizam o mesmo gerador sonoro, embora não seja possível afetar de maneira significativa a qualidade (timbre) da nota produzida. O **SOUND** controla a freqüência e a duração do som, enquanto o **PLAY** permite que você selecione uma seqüência de notas musicais para executar uma melodia, por exemplo. O comando **SOUND** pode ser utilizado para produzir bipes (simples tons sonoros, de freqüência e duração fixas), como mostramos no programa a seguir. Quando rodá-lo, não se esqueça de ligar o volume na sua TV; do contrário, não ouvirá nada!

```
10 FOR Q=1 TO 5
20 SOUND 200,1
30 NEXT Q
```

A série de bipes pode ser alterada mediante a modificação dos números após o comando **SOUND**. O primeiro número controla a freqüência da nota e pode ter um valor de 1 a 255. A nota mais baixa é produzida por 1 e a mais alta por 255 (para alguém que conhece algo sobre música: o valor 89 corresponde à nota dó da escala do meio de um piano).

O segundo número após o comando **SOUND** regula a duração do tom sonoro. Novamente, a gama possível de valores vai de 1 a 255; o valor 16 fornece cerca de um segundo de duração.

Esses bipes podem ser adequados a jogos especiais ou de fantasia, mas não são de muita utilidade quando o tema do jogo é mais realista. Assim como no caso dos efeitos gráficos simplificados para produzir explosões (flashes) descritos anteriormente, deve-se ser criterioso ao utilizá-los. Nunca deixe de empregar efeitos sonoros adequados ao "clima" do jogo.

Quando se precisa de um tipo mais sofisticado de efeito sonoro, é recomendável abandonar o comando **SOUND**, cujos recursos são limitados. Ao invés dele, é preferível utilizar o comando **PLAY**, como será explicado mais adiante.

OUTRA VEZ UM LABIRINTO

Se você gravou o programa de criação de labirintos aleatórios para o TRS-Color (dado na última lição de programação de jogos), poderá melhorar muito o jogo, adicionando os efeitos sonoros a seguir.

Substitua a linha 230 no programa do labirinto pela seguinte:

```
230 IF X<>LX OR Y<>LY THEN PUT
(X1,Y1)-(X1+BS-1,Y1+BS-1),B,PSE
T:LX=X:LY=Y:PLAY"T5005DG"
```

Mude a linha 240 para:

```
240 IF F=1 THEN F=0:SC=SC+(TI-T
IMER):TI=TI-10:PLAY"T1003BCDEFG
A"GOTO 130
```

e substitua a linha 500 por:

```
500 CLS:SCREEN 0,0:PLAY"T302L2C
L4CL12CL2CL4D#L8DL4DL8CL4C01L8B
02L2C":LI=LI-1
```

Agora, rode o programa e escute o que acontece quando o homenzinho se move pelo labirinto. Você obterá um som diferente no momento em que o tesouro for encontrado, e uma "marcha fúnebre" quando o jogador perder uma "vida".

Ao lidar com efeitos sonoros, procure evitar que eles retardem o seu jogo, pois toda vez que produz um som o computador interrompe qualquer outra coisa que esteja fazendo. Em programação de jogos, isso significa que qualquer

movimento na tela cessa enquanto o som estiver sendo produzido. Na verdade, você pode utilizar sons em vez de laços **FOR...NEXT** para introduzir pausas mais ou menos longas nos seus programas. O som na linha 230, por exemplo, é muito curto, pois o espaço de tempo foi reduzido ao mínimo. Trabalhando com sons, você pode até mesmo fazer o ajuste fino de seus programas.

O COMANDO PLAY

O comando **PLAY** opera sempre com base em um cordão alfanumérico, que transporta instruções para o computador. O cordão na linha 230 do programa anterior contém as instruções **T50O5DG**. O **T** significa Tempo (ou velocidade, se você preferir) e pode ser estabelecido em qualquer valor de 1 a 255. A letra **O** é a oitava, e pode assumir um valor de 1 a 5 (1 é o valor mais baixo, e 5 é o mais alto). As duas últimas letras, **D** e **G**, são notas, no sistema americano de notação (C é dó, D é ré, E é mi, etc.). De um modo muito geral, o que **T50O5DG** significa é: "toque duas notas altas rapidamente".

Tente alterar os valores de **T** e **O** para ver que tipos de efeitos sonoros você poderá obter.

Na linha 240, o cordão é **T10O3BCD EFGA**, que executa uma escala crescente.

A marcha fúnebre na linha 500 é mais complexa. Uma explicação detalhada de como escrever música para o computador deverá esperar mais um pouco, mas observe o quanto a duração da nota L é variada durante a melodia. Na verdade, você pode variar qualquer um dos parâmetros em um cordão, em qualquer ponto durante a melodia, tornando assim o comando **PLAY** muito flexível.

EXPLOSÕES

O comando **PLAY** pode igualmente ser utilizado para produzir efeitos sonoros simulando explosões, que serão de grande utilidade na maioria de seus jogos de ação. Os seguintes efeitos sonoros podem ser utilizados sozinhos, ou com as imagens da lição "Bombardeios e Explosões" (páginas 121 a 127).

Tente esse efeito:

```
10 PLAY"T1600L30GF#FE#D#DC#D#D
FE#"
```

Você poderá usá-lo quando algo for atingido ou explodir na tela. Incorpore-o, por exemplo, ao programa da página 64, acrescentando o comando **PLAY** à linha 270, antes do comando **GOTO**.

Um interessante som de reverberação pode ser conseguido assim:

```
10 PLAY"T8001L2BFBFBFBFBFBFBFBF
BF"
```

Na oitava mais baixa, duas notas se repetem sem cessar. Tente manipular a velocidade e a duração das notas para obter variações sobre esse som.

Você poderá conseguir uma repetição de seus efeitos sonoros utilizando uma rotina como a que se segue:

```
10 F$="T10002L10AGBE#DFADF#"
20 F$=F$+F$
30 PLAY F$
```

A linha 10 contém o cordão de instruções para o som. A linha 20 concatena o cordão, de modo que, quando a linha 30 o executa, o som se repete.

UMA SIRENE

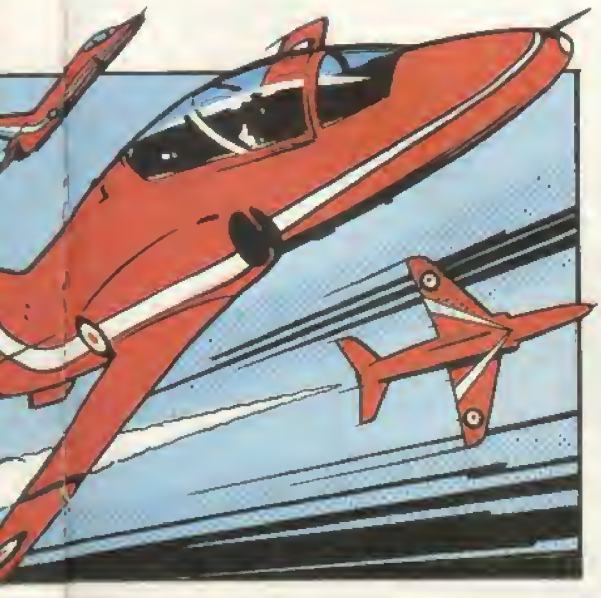
A repetição de um som tem outras aplicações. Se você deseja obter o ruído de uma sirene, empregue um programa como esse:

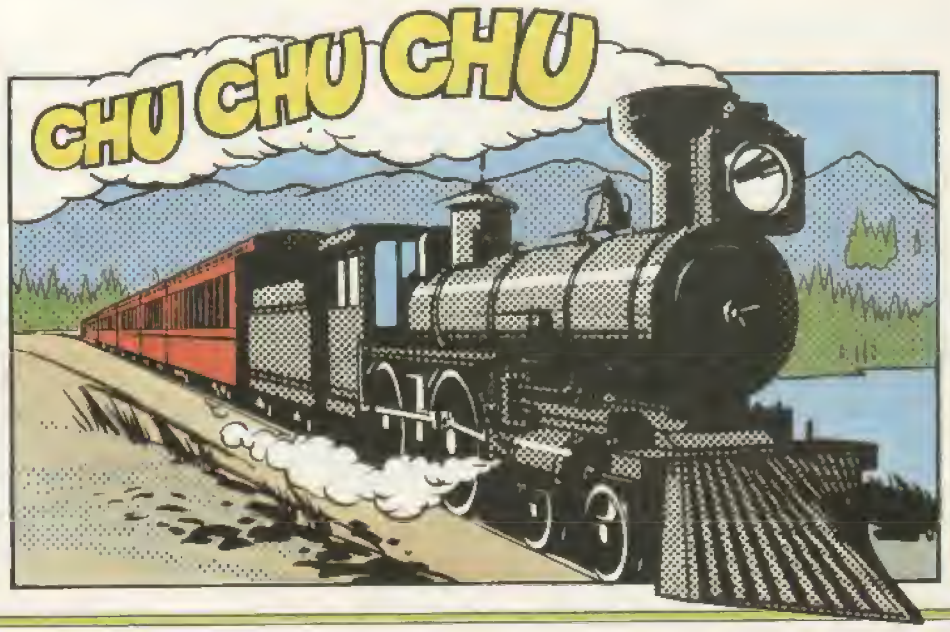
```
10 CLEAR 250
20 F$="T15004L8CDD#EF#FGF#FED#D
C#CD#DC#C"
30 F$=F$+F$+F$+F$
40 PLAY F$
```

Para se obter o efeito de sirene, constrói-se um longo cordão, concatenando-se quatro partes do **F$** original, antes de executá-lo. O computador TRS-Color oferece, ao ser ligado, uma quantidade limitada de espaço para variáveis alfanuméricas, de modo que o cordão que contém o novo efeito sonoro será muito maior do que a memória disponível.

Portanto, a linha 10 é usada para reservar mais espaços para variáveis alfanuméricas, por intermédio do comando **CLEAR 250**, que estipula 250 bytes para esse fim. Tenha cuidado em reservar os 250 bytes extras de memória quando você utilizar o efeito sonoro em um programa de jogos; caso contrário, você obterá uma mensagem de erro: OS — *out of string space*.

Não fique desapontado se os seus experimentos não produzirem sons comparáveis aos melhores efeitos em jogos comerciais. Alguns efeitos muito bons são possíveis em BASIC, mas os mais sofisticados são escritos em código de máquina — um tópico que será abordado futuramente em INPUT.





MEMÓRIAS SÃO FEITAS ASSIM

Dividida em RAM e ROM, a memória do computador armazena dados, resultados e programas. Quase tão complexa quanto a memória humana, dela depende o funcionamento da máquina.

A memória dos micros é constituída por milhares de minúsculos circuitos integrados de silício que podem ser ligados e desligados individualmente. Esses circuitos são organizados, dentro do chip, em grupos de oito. Cada grupo representa um byte, isto é, um número binário de oito bits, ou o correspondente a dois dígitos em hexadecimal. Internamente, tal número é usado para definir um endereço para cada locação de memória.

O ESPAÇO DE MEMÓRIA

Em computação, um K é aproximadamente análogo ao k (quilo) do sistema métrico. Na realidade, 1K em computação equivale ao número hexadecimal 400, que é igual a 1 024 em decimal. Assim, se quisermos identificar 64K de memória, devemos numerá-los de 1 a 65 536 em decimal (ou, de 0000 a FFFF, em hexa).

O número hexadecimal de quatro dígitos atribuído a uma locação de memória é conhecido como o seu endereço.

Os micros aqui citados podem atingir uma memória interna de 64K, pois contam com um espaço de endereçamento de dezesseis bits. Entretanto, não é esta a quantidade habitual de memória anunciada pelos fabricantes. O modelo mais comum do Spectrum (por exemplo, o TK-90X, no Brasil) tem 48K; o TRS-Color, 32K; e o Apple II e o TRS-80, 48K cada um. De fato, porém, todos eles têm 64K de espaço de memória, ao todo. Esse número refere-se apenas à quantidade de memória que o usuário ou o programador podem utilizar: os outros 32 ou 16K, conforme o caso, são reservados para utilização pela própria máquina. Nos micros citados, os fabricantes numeram as locações de memória de 0000 a FFFF. O Spectrum ou o ZX-81 de 16K (os menores) — que têm na realidade 32K de memória ao todo — numeram suas locações de memória de 0000 a 7FFF.

ROM E RAM

Além da organização básica em bits e bytes, o espaço de memória do computador é dividido em subunidades: as *páginas*, definidas como *conjuntos endereçáveis* de 100 locações de memória em hexa, cada uma (256 em decimal).

A *página 0 (zero)* vai de 0000 a 00FF; a *página 1 (um)*, de 0100 a 01FF, e assim por diante. A organização em páginas facilita a programação do sistema, pois páginas específicas podem ser reconhecidas pelo hardware da UCP.

Existem dois tipos de memória interna: ROM e RAM. ROM (*Read-Only Memory*) significa *memória somente de leitura*. Ou seja, pode-se ler o conteúdo das locações de memória ROM mas não se pode escrever nelas. Protegida contra alterações externas, a informação contida é fixada permanentemente quando os chips ROM são produzidos.

A memória ROM guarda as instruções operacionais e o interpretador que traduz seus programas BASIC para código de máquina. Os códigos contidos na ROM também são responsáveis pela imposição de uma estrutura ao resto da memória. Aberta ao usuário, a RAM (*Random Access Memory*) significa *memória de acesso aleatório*. Algumas locações na RAM são controladas pela ROM: se você tentar escrever nessas lo-

- DE QUE É FEITA A MEMÓRIA
- QUANTA MEMÓRIA O SEU COMPUTADOR POSSUI?
- MEMÓRIAS ROM E RAM
- O QUE CONTÉM A MEMÓRIA?
- ONDE OS PROGRAMAS BASIC SÃO ARMAZENADOS
- COMO O COMPUTADOR ARMAZENA NÚMEROS GRANDES
- PONTEIROS E INDICADORES

cações reservadas de memória, os programas gravados na ROM se encarregarão de fazê-las voltar ao seu conteúdo original. Apesar disso, RAM é uma espécie de folha em branco, na qual você pode escrever o que quiser.

PONTEIROS E INDICADORES

Os mapas de memória mostrados adiante são uma representação gráfica da memória RAM e ROM de cada tipo de micro. As áreas representadas, contudo, não estão em sua posição física exata, pois o espaço da memória é dividido em diferentes chips. Alguns dos limites entre as seções da memória — como a fronteira entre a ROM e a RAM — coincidem com a mudança de um chip para outro. Outros, porém, são flexíveis e suas posições são indicadas por um *ponteiro* ou *indicador* na área de variáveis do sistema.

Um ponteiro é uma locação da memória (ou melhor, um par de locações da memória) que armazena o endereço de uma outra locação — neste caso, o início de uma seção particular da memória. O endereço de qualquer byte de memória é constituído por um número binário de dois bytes de comprimento (dezesseis bits), de modo que precisa ser armazenado em duas locações adjacentes de memória.

A memória ROM de 16K do Spectrum vai de 0000 a FFFF e contém o interpretador e o editor de programas em BASIC, bem como várias rotinas de entrada e de saída e o conjunto de caracteres que guardam os dados para as letras do alfabeto, números e outros símbolos gráficos. Os 48K restantes — que vão de 4000 a FFFF — ou os 16K restantes — que vão de 4000 a FFFF — são memória RAM.

As funções das áreas em que a memória RAM está dividida são as seguintes:

A *área de exibição* controla o que é mostrado no vídeo. Cada locação da memória corresponde a uma linha de oito pixels.

A *área de atributos* controla as cores de fundo **(PAPER)** e de frente **(INK)** das 768 locações de caracteres na tela, e o tipo de exibição a ser feita (intensidade normal, brilhante ou piscante).

A *memória intermediária da impressora* retém a próxima linha do texto que será enviada para a impressora.

A *área de variáveis do sistema* contém as locações que guardam os endereços do início das áreas especificadas acima (apontadores).

A *área de mapeamento de microdrives* existe apenas nas unidades de microfita (*microdrives*) para o Spectrum, conectadas ao computador. Em caso contrário, o apontador *CHANS*, cujo endereço é armazenado nas locações 23 631 e 23 632 (5C4F e 5C50, em hexadecimal), é deslocado para 5BC6.

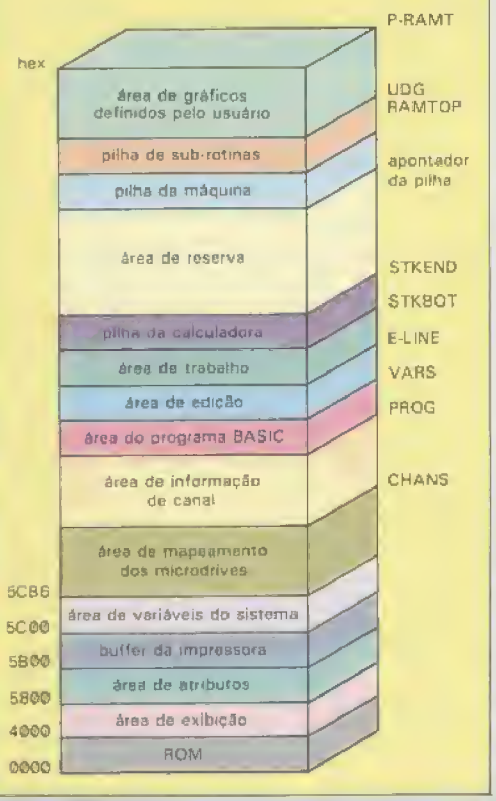
A *área de informação de canal* transporta os dados de entrada e de saída. Ela conduz a entrada do teclado para a parte mais baixa da tela de TV e a saída do programa para o restante da tela, para o espaço de trabalho mais acima na memória e para uma impressora.

A *área do programa BASIC* retém as linhas de qualquer programa BASIC digitado ou carregado; seu tamanho depende da extensão do programa. Inicia-se no endereço fornecido pela variável de sistema **PROG**, que é armazenada nas locações 23 635 e 23 636 (5C53 e 5C84) na área das variáveis do sistema. Isto apontará para a locação 23 755 (5CCB em hexa), se nenhum microdrive estiver conectado.

A *área das variáveis* armazena os valores das variáveis que estão sendo utilizadas no programa BASIC em curso. Começa na locação apontada por **VARS**, armazenado nas locações 23 627 e 23 628 (5C4B e 5C4C em hexa) da *área das variáveis do sistema*. Quando um programa é rodado, o início da área das variáveis permanece onde está, mas o final vai crescendo para cima, à medida que novas variáveis são definidas pelo programa em BASIC.

A *área de edição* é o local onde é fei-

MAPA DE MEMÓRIA DO SINCLAIR SPECTRUM



ta a edição das linhas BASIC. Quando o <**ENTER**> é apertado, as linhas são copiadas para a área do programa BASIC. A área de edição começa na **E-LINE**, cujo endereço é retido nas locações 23 641 e 23 642 (5C59 e 5C5A em hexa) na área das variáveis do sistema.

O *espaço de trabalho* serve para tarefas gerais, como a armazenagem dos dados de entrada e a concatenação de strings. O **WORKSP** é armazenado em 23 649 e 23 650 (5C61 e 5C62 em hexa) na área das variáveis do sistema.

A *pilha de cálculo* é utilizada para reter números em ponto flutuante, números inteiros de cinco bytes e o conjunto de parâmetros de cinco bytes. Começa em **STKBOT**, cujo endereço ocupa as locações 23 651 e 23 652 (5C63 e 5C64 em hexa), e termina em **STKEND**, que é encontrado em 23 653 e 23 654 (5C65 e 5C66 em hexa).

A *área de memória de reserva* permite que as áreas da memória situadas acima e abaixo dela cresçam até que **STKEND** encontre o indicador de pilha.

Acima dos bytes sobressalentes está a *pilha da máquina,* empregada pela UCP quando um programa BASIC é rodado.

A *pilha de sub-rotinas* (**GOSUB**) armazena os números de linha para os quais um programa deve retornar quando completar cada sub-rotina chamada.

O ponteiro **RAMTOP** indica o final da memória RAM disponivel para o armazenamento de programas. Seu endereço é retido em 23 730 e 23 731 (5CB4 e 5CB5 em hexa) na área das variáveis.

Acima de **RAMTOP** há 168 locações de memória que podem ser usadas para armazenar 21 representações ou padrões gráficos definidos pelo usuário. Entretanto, o topo da RAM é armazenado em uma variável do sistema e pode ser deslocado para uma posição mais abaixo na memória, usando-se o espaço de reserva para as pilhas de sub-rotina e de máquina. Isto pode ser feito através de programação em código de máquina. Normalmente, um programa em código de máquina fica acima do **RAMTOP** abaixado.

O apontador **P-RAMT** assinala o *topo físico* da memória RAM; isto é, acima desse ponto não existem mais locações da memória nos chips do Spectrum. O **P-RAMT** é uma variável do sistema, com endereço armazenado nas locações 23 732 e 23 733 (5CB5 e 5CB6 em hexa).

Para verificar o valor de **P-RAMT** ligue a máquina e entre essa linha:

```
PRINT PEEK 23732+256*PEEK 23733
```

Isto retornará o valor 65535 no Spectrum de 48K, ou o valor 32767 no modelo de 16K. Se nenhum destes valores for retornado pela linha, então alguma coisa está errada com o seu computador. Pode-se examinar o conteúdo dos outros apontadores na área de variáveis do sistema, utilizando a mesma linha **PRINT**, apenas substituindo os valores correspondentes às duas locações de memória da variável.

A palavra-chave **PEEK** em BASIC "olha" os conteúdos dos bytes da memória e retorna o equivalente decimal do número que encontrar. O endereço tem o comprimento de dois bytes (requer duas locações da memória para ser armazenado). O Spectrum divide o endereço hexa de quatro dígitos — por exemplo, o endereço normal FF57 de **RAMTOP** — em duas partes de F e 57.

O byte mais baixo, 57, vai dentro do endereço menor, e o byte mais alto, FF, vai no endereço maior. Isto explica por que a segunda das locações da

memória no programa acima é multiplicada por 256; do contrário, o Spectrum poderia retornar ao equivalente decimal de FF, ao invés de FF00.

Os micros compatíveis com o modelo Sinclair ZX-81 têm 8K de ROM, com endereços que vão de 0000 a 1FFF em hexa, ou de 0 a 8 190 em decimal. Os outros 8K de memória, de 2000 a 3FFF, não são utilizados.

O mapa de memória mostrado aqui serve para um ZX-81 com um cartucho de expansão de 16K RAM, cujos endereços vão de 4000 a 7FFF em hexa, ou 16 384 a 32 767 em decimal. No caso de o computador ter um cartucho de RAM de 8K ou um de 4K, basta "achatar" as áreas de memória no espaço correspondente de 8K ou 4K, e fazer com que o topo físico da memória RAM passe a ser 5FFF (24 575 em decimal) para o cartucho de 8K, ou 4FFF (20 479 em decimal), para o cartucho de 4K.

Em todos esses casos, as *variáveis do sistema* ocupam a área de 4000 a 4087 em hexa, ou 16 384 a 16 509 em decimal. As outras áreas não são fixas, e os endereços correspondentes aos seus limites — **D-FILE, WARS, E-LINE, STKBOT, STKEND, ERR-SP** e **RAMTOP** — são armazenados como apontadores nessa área das variáveis do sistema.

A *área de programa BASIC*, que vai de 16 509 a **D-FILE**, contém o programa em BASIC do usuário. Da locação **D-FILE** até a locação **WARS**, fica o *arquivo de exibição*.

A *área de variáveis* vai se expandindo à medida que um programa em BASIC é executado e novas variáveis são definidas em seu interior. Essa área tem o seu final assinalado pela locação da memória que contém 80, em hexa.

Entre as locações **E-LINE** e **STKBOT** estão a *área de edição* e o *espaço de trabalho*. A linha que está sendo digitada pelo teclado ocupa a área de edição. Seu conteúdo é transferido para a área do programa assim que a tecla <**ENTER**> é pressionada.

Entre a locação **STKEND** e o indicador da pilha de máquina existe uma área de reserva de memória, para onde as áreas acima e abaixo dela podem se expandir. A *pilha de máquina é usada* pela máquina quando um programa BASIC é rodado; a *pilha de sub-rotinas* (**GOSUB**) armazena os números de linha para os quais um programa deve retornar depois da execução de uma sub-rotina.

O apontador RAMTOP designa o topo físico da memória — isto é, 32 767, 24 575, 20 479 ou 17 407, dependendo do cartucho de expansão RAM conectado à máquina. Tecle a linha seguinte:

```
PRINT PEEK 16388+256*PEEK 16398
```

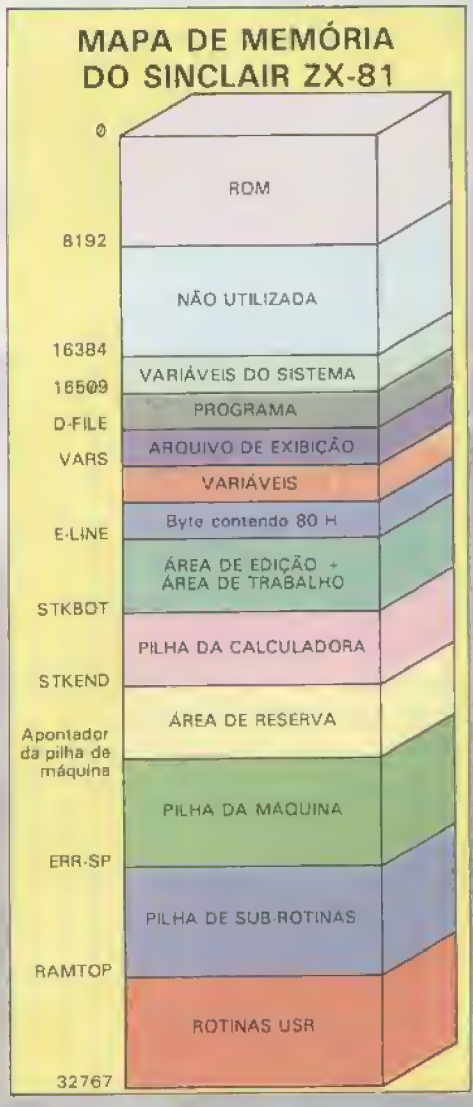
A linha retorna o valor decimal correspondente à RAM máxima disponível no sistema. Você pode examinar o conteúdo dos outros apontadores na área de variáveis do sistema, utilizando a mesma linha **PRINT** e substituindo os valores correspondentes às duas locações de memória da variável.

A palavra-chave **PEEK** em BASIC "olha" os conteúdos de cada byte da memória e retorna o equivalente decimal do número que encontrar.

O endereço tem o comprimento de dois bytes. O ZX-81 divide o endereço hexa de quatro dígitos — por exemplo, o endereço normal FF57 de RAMTOP — em duas partes de FF e 57. O byte mais baixo, 57, vai no endereço menor, e o byte mais alto, FF, no endereço maior. Isto explica por que a segunda das duas locações da memória no programa acima é multiplicada por 256.

O TRS-Color tem uma memória ROM de 32K, que vai da locação &H8000 à &HFFFF. Essa área é dividida em duas outras: uma que controla a entrada e a saída do computador em relação ao cartucho de ROM removível, e outra, não removível, que contém o sistema operacional e o interpretador BASIC.

Os programas em BASIC e suas variáveis são armazenados na memória RAM, entre as locações &H3600 a &H7FFF. Existe também uma *pilha* que ocupa a parte mais elevada de RAM, logo abaixo da ROM, de &H7FFF para baixo. Mas a pilha pode descer um pouco mais, de modo a criar espaço para um programa em código de máquina. Isto é feito alterando-se o valor da variável do sistema, que aponta para a base da pilha.



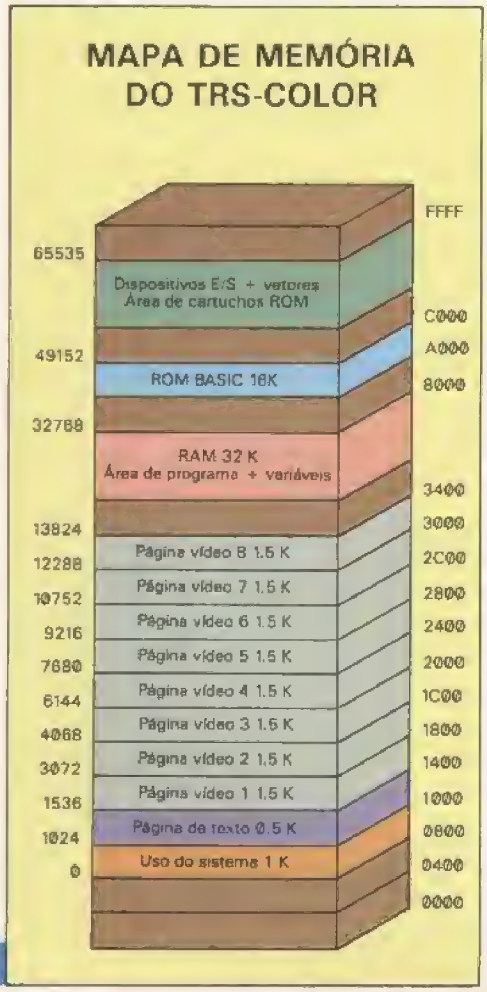
Uma parte da RAM é reservada para oito *páginas de gráficos* de alta resolução, que não correspondem exatamente às páginas de memória mencionadas antes. Estas contêm 256 locações, ou 0.25K. As páginas de gráficos, ao contrário, têm 1.5K, e cada uma guarda informação gráfica suficiente para preencher toda a tela na modalidade de resolução mais baixa — **PMODE 0** —, enquanto a modalidade de resolução mais alta — **PMODE 4** — necessita quatro dessas páginas para preencher totalmente a tela. O computador reserva automaticamente quatro páginas, mas **PCLEAR** pode ser usado para reservar mais.

Quando não estão em uso, essas páginas são deixadas vazias. Mas, se uma memória extra for requisitada, elas poderão ser limpadas até a página 1 do BASIC. Isto permite o acesso a uma memória extra de 10.5K.

A *tela de texto* ocupa 1/2K entre &H400 e &H600. Isto corresponde a uma locação de memória para cada espaço de caracteres na tela de dezesseis linhas de 32 caracteres. Os caracteres, formados por 96 pontos na tela (oito por doze pixels), são gerados por um *chip gerador de vídeo* separado.

As *variáveis do sistema* estão armazenadas na área que vai de &H000 a &H400. São uma coleção de indicadores ou apontadores, que fornecem os endereços do início de diversas áreas na memória e de outras variáveis do sistema.

O primeiro grupo de 256 bytes — de &H00 a &HFF — é conhecido, no TRS-Color, como *página direta*. Qualquer das páginas de memória pode ser designada como página direta, ou seja, a página cujas locações podem ser endereçadas mediante a utilização de um endereço curto de um byte, em vez do endereço normal e maior de dois bytes. Isto é feito definindo-se o *registro da página direta* no microprocessador.



O TRS-80 tem um espaço máximo de memória RAM de 48K, quando usado com o BASIC Nível II (máquinas com gravador cassete), ou com o TRSDOS e equivalentes (máquinas com disquete). Quando esses micros são usados com o sistema operacional CP/M, essa memória pode ser expandida internamente para 64K de RAM. Trataremos aqui apenas do mapa de memória de máquinas compatíveis com o modelo III da Radio Shack, como é o caso do Prológica CP-500.

O espaço total de memória interna do TRS-80 é de 65 535, ou HFFFF, em hexa. O espaço que vai do endereço 0 a 14 435 (ou H0000 a H3800) é ocupado pela memória ROM do sistema: 12K de EPROM contendo o sistema operacional e o interpretador BASIC Nível II, e 2K de EPROM adicionais para uso do sistema (total de 14K de ROM).

A memória RAM começa sempre na locação 14 336 e se estende até o topo físico da memória, que pode ser 32 767 (máquinas com 16K), 49 151 (máquinas de 32K) ou 65 535 (máquinas de 48K). Os endereços em hexa são: H7FFF, HBFFF e HFFFF.

A RAM é dividida, por sua vez, em duas partes principais: uma reservada para uso do sistema, que vai de 14 336 a 17 384 (H3800 a H43E8) e outra, que começa em 17 385 (H43E9) e vai até o topo lógico da memória (**RAMTOP**). Essa locação está armazenada na área de variáveis do sistema, e é definida pela primeira vez quando o computador é ligado, e faz a pergunta ao operador (MEM SIZE?).

A área do sistema é subdividida, por sua vez, em diversas áreas de comprimento fixo e uma de participação flexível. As áreas de comprimento fixo são as seguintes (veja o gráfico):

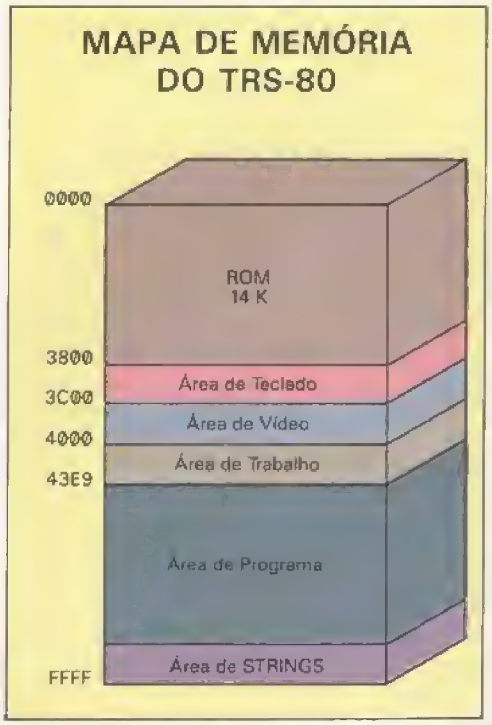
A *área matricial do teclado* contém as locações de memória mapeadas no teclado. Qualquer tecla pressionada "liga" bytes correspondentes nessa área (que vai de 14 336 a 15 359).

A *memória de vídeo* também corresponde a um mapeamento direto das 1 024 posições de caracteres na tela, em memória RAM. Estende-se de 15 360 (canto superior esquerdo da tela) a 16 383 (canto inferior direito). Assim, uma posição N na tela (correspondente ao parâmetro do <**PRINT**>) é armazenada na posição absoluta de RAM igual a **15.360+N**.

A área que vai das locações 16 384 a 17 384 (mil bytes, portanto) é a *área do sistema*, que contém apontadores, *buffers*, indicadores e outras locações especiais de uso particular do interpretador BASIC e que podem ser examinados e alterados pelo usuário por meio de comandos **PEEK** e **POKE**. O conteúdo inicial dos apontadores e indicadores é estabelecido quando a máquina é ligada, ou quando é dado um RESET de iniciação.

A *área de programas e dados* começa na locação 17 385 (H3E9). Tem três diferentes compartimentos: a área do programa, a das variáveis e a de cordões alfanuméricos. Os limites entre essas três subáreas são alterados dinamicamente, em função da execução do programa em BASIC e de alguns de seus comandos. Por exemplo, o comando **CLEAR** fixa o tamanho máximo da área de cordões, logo abaixo do topo lógico da RAM. Ao ser introduzido, o interpretador aloca automaticamente cinqüenta bytes a essa área.

O procedimento usual para colocar um programa em linguagem de máquina do usuário é fixar um limite lógico máximo de RAM, seja através da pergunta inicial, seja através da manipulação do ponteiro que contém o endereço. Isto cria uma área



protegida de memória onde é colocado o programa em linguagem de máquina por meio do monitor residente do TRS-80/III ou de programas monitores e Assembler carregados externamente.

Os micros MSX apresentam algumas diferenças entre si. Por isso, tomaremos como exemplo o micro nacional Sharp HB-8000 (Hot-bit). Este dispõe de um espaço máximo de memória de 128K, dividido em quatro *slots,* ou encaixes disponíveis, quando usado com o BASIC para gravador cassete. Essa memória tem RAM disponível de 64K.

O espaço de memória do Sharp é dividido em dois compartimentos de 65 535 bytes, ou &HFFFF, em hexa. No primeiro, o espaço que vai de 0 a 32 767 (ou &H0000 a &H7FFF) é ocupado pela memória ROM; ele contém o interpretador BASIC e o sistema operacional. O espaço restante, de 32 768 a 65 575 (&H8000 a &HFFFF), é utilizado para endereçamento dos cartuchos removíveis de ROM.

A memória RAM básica está no segundo compartimento. A área que vai de &H0000 a &H7FFF (*área livre*) não é controlada pelo interpretador BASIC.

A segunda área da RAM é controlada pelo interpretador BASIC e é dividida em duas partes principais: a primeira, mais alta e de comprimento fixo, é reservada para uso do sistema e vai de &F380 até o topo físico da RAM. É a *área do sistema* e contém apontadores, *buffers,* indicadores e outras locações de uso particular do interpretador BASIC e que podem ser examinados e alterados pelo usuário (comandos **PEEK** e **POKE**).

A segunda área, mais baixa, contém tudo o que é de uso do programa BASIC armazenado e é subdividida em áreas de comprimento variável.

A *área de blocos de controle dos arquivos* tem como limite máximo o topo lógico da área do usuário. Esse valor é definido como sendo &HF380, mas pode ser modificado para baixo (comando **CLEAR**). Se um valor diferente deste for estipulado pelo **CLEAR,** a área livre entre essa locação máxima e a locação &HF380 fica disponível para abrigar programas em código de máquina. A área dos blocos de controle contém *buffers* e indicadores para transferências de entrada e saída. Seu tamanho é definido internamente ou modificado pelo comando **MAXFILES** do BASIC.

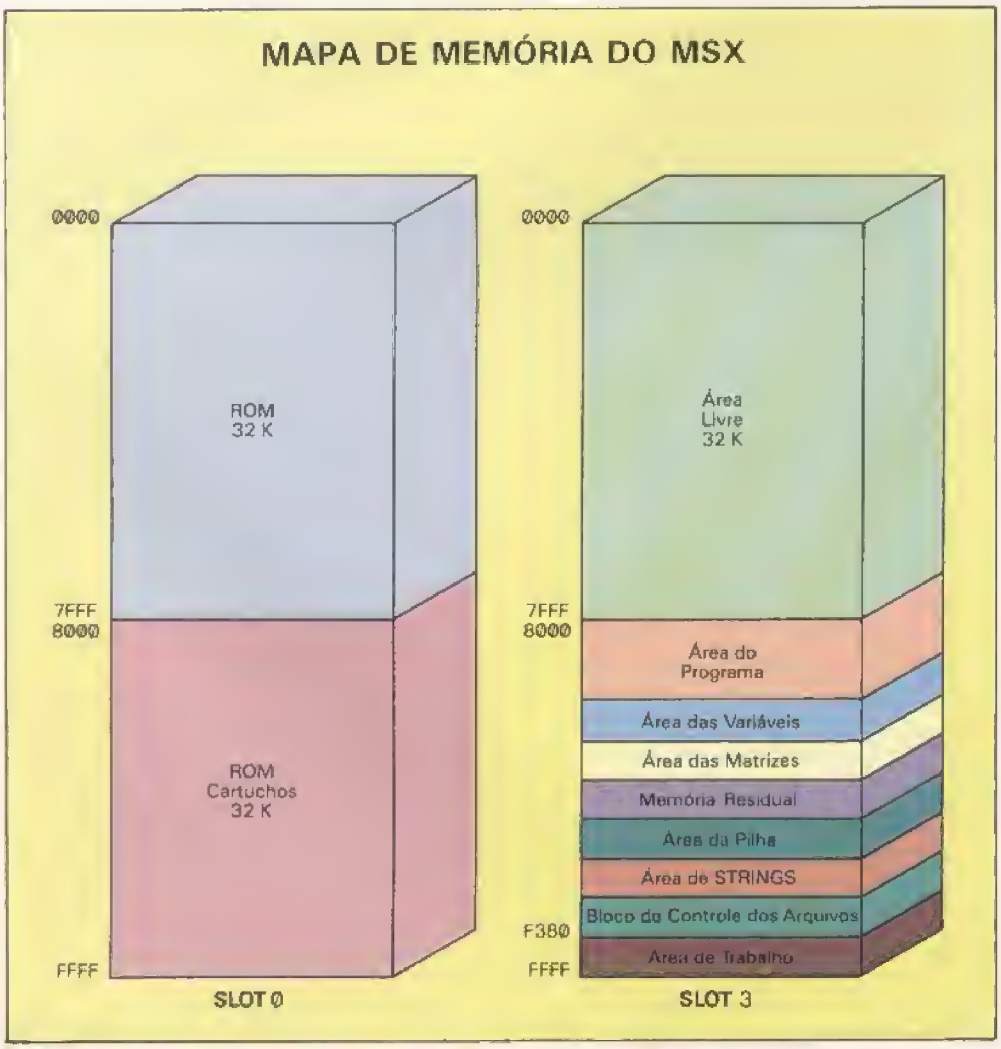
A *área de cordões alfanuméricos* vem logo abaixo e contém os cordões definidos durante a execução do programa. Tem tamanho estabelecido pelo comando **CLEAR.** Este pode conter um segundo argumento, que é a definição do topo lógico da RAM. Por exemplo, **CLEAR 200, &HDFFF** fixa esse valor em &HDFFF.

A seguir vem a *área da pilha,* que tem comprimento fixo e contém os endereços de retorno das sub-rotinas (**GOSUB**) e dos laços **FOR...NEXT.**

A área que se inicia em 32 768 (&H8000) é a *área de programa,* que abriga o texto do programa. Acima dela existem duas áreas dinâmicas, que crescem à medida que o programa é executado: a *área de variáveis* e a *área das matrizes.* A área de reserva de memória para programa (*área residual*) corresponde à diferença entre a soma das áreas fixas (pilha, cordões e blocos de E/S) e a soma das áreas variáveis (programa, variáveis e texto).

A *memória de vídeo* do MSX também corresponde a um mapeamento direto de posições de caracteres ou de gráficos em diversas páginas. São 16K de RAM que têm seu endereçamento à parte.

A tela de vídeo é tratada pelo microprocessador central como um dispositivo de entrada e saída, e o BASIC dispõe de vários comandos para fazer funcionar esse esquema, inclusive leitura e gravação absoluta (**VPEEK** e **VPOKE**).



O Apple II e o TK-2000 utilizam o microprocessador 6502, de oito bits. Seu espaço total de endereçamento de memória é de 64K, dividido em 256 páginas de 256 bytes. Essas páginas são alocadas para diferentes funções pela UCP e pelas rotinas operacionais, e constituem o mapa de memória do Apple. Na configuração padrão desses micros, somente 48K são disponíveis para o usuário (RAM), mesmo assim parcialmente, pois uma parte é reservada para uso do sistema. Os 16K restantes correspondem à memória ROM. A forma básica de organização da memória é semelhante para os micros da linha Apple II e para o TK-2000.

A organização da memória do Apple II depende do tipo de linguagem empregada (INTEGER BASIC, FPBASIC ou APPLESOFT), de se está sendo utilizado disquete ou não, e da versão do sistema operacional de disquete (DOS). Além disso, é

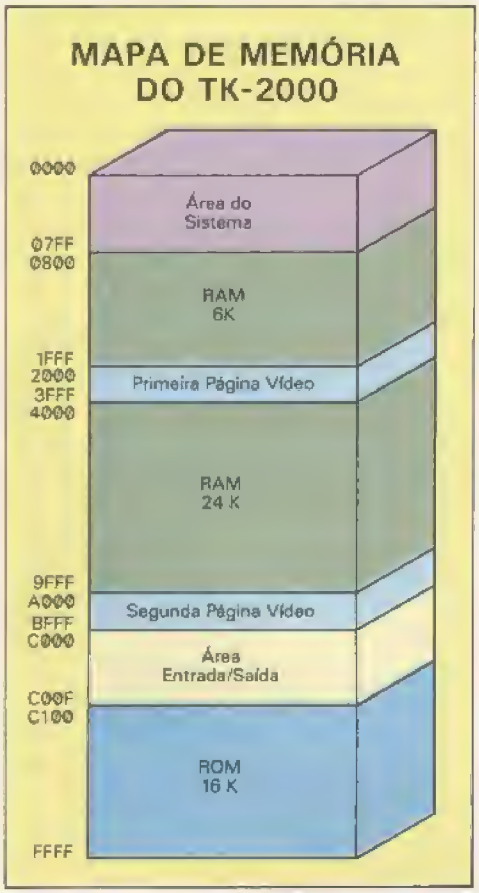
possível expandir-se a memória RAM mediante bancos adicionais, para 64K ou 128K. Examinaremos aqui o mapa de memória para a configuração básica, ou seja, um Apple II com 48K de memória RAM e sistema cassete:

A *área da memória ROM,* que se estende das páginas **$C1** a **$FF** (em hexa), contém o código de máquina necessário para a operação e programação da máquina — o interpretador BASIC, o monitor, e, no caso do TK-2000, também um disassembler e um mini-assembler.

A *área de entrada/saída* é uma página de 1K, que inclui uma série de registros, áreas de memória intermediária, apontadores e indicadores relativos aos dispositivos de entrada e saída da máquina (vídeo, som, gravador cassete, teclado, seleção de RAM, etc). Além disso, existem áreas de memória reservadas para as *páginas de vídeo,* que servem para mapear os textos e gráficos que nele aparecem.

Os endereços das áreas de vídeo são diferentes para o TK-2000 e para os micros da linha Apple:

	Apple	TK-2000
TEXTO	Pág. 1	0400-07FF 2000-3FFF
TEXTO	Pág. 2	0800-0BFF A000-BFFF
HGR	Pág. 1	2000-3FFF 2000-3FFF
HGR	Pág. 2	4000-5FFF A000-BFFF



O Apple tem quatro áreas de vídeo: duas páginas situadas na parte baixa da memória (antes do início da área de programas) e duas na parte alta da RAM. Já o TK-2000 tem somente duas áreas, ambas na parte alta da memória.

A área de texto do Apple e do TK-2000 também é usada para gráficos de baixa resolução (GR), ao passo que as áreas de gráficos de alta resolução são separadas apenas no Apple.

Finalmente, a terceira área é a *área de memória RAM,* que vai da página 0 à página 191 ($00 a $BF, em hexa).

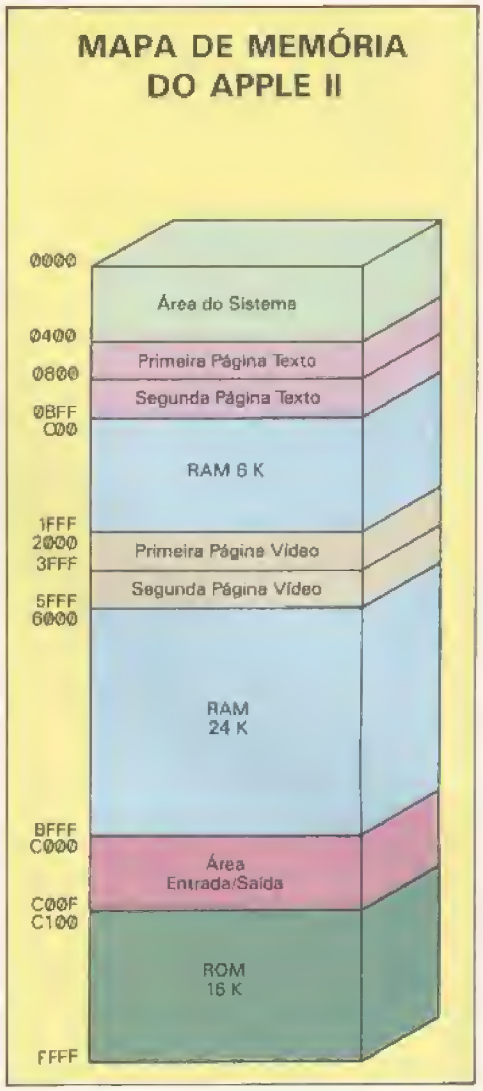
A memória RAM também é subdividida em áreas de tamanho fixo ou variável, conforme a aplicação:

A *área de variáveis do sistema* é armazenada na página 0 e contém apontadores e indicadores atualizados pela UCP. A *pilha da máquina* ocupa a página 1, e tem 256 bytes. Ela serve para diversos fins, podendo inclusive ser utilizada pelo programador em linguagem de máquina; normalmente, ela guarda os endereços de retorno de sub-rotinas e laços em BASIC. A *área de edição,* alocada na página 3, contém o *buffer* (memória intermediária) do teclado, ou seja, a última linha digitada.

A *área de trabalho,* que se estende das páginas 3 a 7, também é reservada para o sistema operacional, para o monitor e para diversas rotinas de E/S. Alguns endereços dessa área podem ser utilizados pelo programador em linguagem de máquina para colocar seus próprios programas, bem como para modificar alguns ponteiros e indicadores que permitem alterar a maneira pela qual a UCP processa os dados na memória e nos dispositivos de E/S.

Finalmente, o espaço que vai da página 8 em diante é a *área do usuário.* Os programas em BASIC começam normalmente na página 8, que também pode ser utilizada para programas em linguagem de máquina. Nos micros da linha Apple, contudo, esse espaço de RAM não fica inteiramente disponível para os programas do usuário. Bem no meio da área do usuário estão as duas páginas de vídeo referidas acima. Usando comandos **POKE,** nos micros da linha Apple, e os comandos **MP** e **MA,** no TK-2000, é possível comutar-se a utilização da primeira área de vídeo, passando-a para a segunda, situada mais acima.

Os programas em BASIC começam a partir de &H0800. Por isso, sobram cerca de seis Kbytes até o começo da primeira área de vídeo. Se você quiser utilizá-la, o truque acima não funcionará. Mas há uma alternativa. A *área de variáveis* do BASIC tem dois limites definidos por dois apontadores na área de variáveis do sistema: **LOMEM,** que define o limite inferior dessa área, e **HIMEM,** seu limite superior. Existem comandos equivalentes em BASIC para alterar esses endereços. Assim, deslocando-se simultaneamente o endereço, de início de um programa BASIC para uma posição *acima* da primeira área de vídeo (isto é feito com dois comandos **POKE**) e os parâmetros **HIMEM** e **LOMEM,** é possível preservar-se a primeira página de vídeo.



O Apple II e o TK-2000 têm a capacidade de selecionar a RAM ou a ROM para execução de programas residentes em memória. Essa opção é armazenada em dois indicadores situados na área de entrada/saída, que podem ser modificados por dois **POKE.** No TK-2000, o apontador **RO** seleciona a ROM e está na locação **$C05A.** O Apontador **RA** seleciona a RAM e está em **$C05B.** Ao ser ligada a máquina, é selecionada a ROM, o que ativa o interpretador BASIC residente. O programador pode alterar esses endereços e dar ordem para que o micro execute código de máquina (primeiro endereço da RAM). O botão **<RESET>** e as teclas **<CTRL><RESET>** fazem uma resseleção da ROM.

LINHA	FABRICANTE	MODELO
Apple II +	Appletronica	Thor 2010
Apple II +	CCE	MC-4000 Exato
Apple II +	CPA	Absolutus
Apple II +	CPA	Polaris
Apple II +	Digitus	DGT-AP
Apple II +	Dismac	D-8100
Apple II +	ENIAC	ENIAC II
Apple II +	Franklin	Franklin
Apple II +	Houston	Houston AP
Apple II +	Magnex	DM II
Apple II +	Maxitronica	MX-2001
Apple II +	Maxitronica	MX-48
Apple II +	Maxitronica	MX-64
Apple II +	Maxitronica	Maxitronic I
Apple II +	Microcraft	Craft II Plus
Apple II +	Milmar	Apple II Plus
Apple II +	Milmar	Apple Master
Apple II +	Milmar	Apple Senior
Apple II +	Omega	MC-400
Apple II +	Polymax	Maxxi
Apple II +	Polymax	Poly Plus
Apple II +	Spectrum	Microengenho I
Apple II +	Spectrum	Spectrum ed
Apple II +	Suporte	Venus II
Apple II +	Sycomig	SIC I
Apple II +	Unitron	AP II
Apple II +	Victor do Brasil	Elppa II Plus
Apple II +	Victor do Brasil	Elppa Jr.
Apple IIe	Microcraft	Craft IIe
Apple IIe	Microdigital	TK-3000 IIe
Apple IIe	Spectrum	Microengenho II
MSX	Gradiente	Expert GPC-1
MSX	Sharp	Hotbit HB-8000
Sinclair Spectrum	Microdigital	TK-90X
Sinclair Spectrum	Timex	Timex 2000
Sinclair ZX-81	Apply	Apply 300
Sinclair ZX-81	Engebras	AS-1000
Sinclair ZX-81	Filcres	NEZ-8000
Sinclair ZX-81	Microdigital	TK-82C
Sinclair ZX-81	Microdigital	TK-83
Sinclair ZX-81	Microdigital	TK-85
Sinclair ZX-81	Prologica	CP-200
Sinclair ZX-81	Ritas	Ringo R-470
Sinclair ZX-81	Timex	Timex 1000
Sinclair ZX-81	Timex	Timex 1500
TRS-80 Mod. I	Dismac	D-8000
TRS-80 Mod. I	Dismac	D-8001/2
TRS-80 Mod. I	LNW	LNW-80
TRS-80 Mod. I	Video Genie	Video Genie I
TRS-80 Mod.III	Digitus	DGT-100
TRS-80 Mod.III	Digitus	DGT-1000
TRS-80 Mod.III	Kemitron	Naja 800
TRS-80 Mod.III	Prologica	CP-300
TRS-80 Mod.III	Prologica	CP-500
TRS-80 Mod.III	Sysdata	Sysdata III
TRS-80 Mod.III	Sysdata	Sysdata Jr.
TRS-80 Mod.IV	Multix	MX-Compacto
TRS-80 Mod.IV	Sysdata	Sysdata IV
TRS-Color	Codimex	CS-6508
TRS-Color	Dynacom	MX-1600
TRS-Color	LZ	Color 64
TRS-Color	Microdigital	TKS-800
TRS-Color	Prologica	CP-400

FABRICANTE	MODELO	PAÍS	LINHA
Appletronica	Thor 2010	Brasil	Apple II +
Apply	Apply 300	Brasil	Sinclair ZX-81
CCE	MC-4000 Exato	Brasil	Apple II +
CPA	Absolutus	Brasil	Apple II +
CPA	Polaris	Brasil	Apple II +
Codimex	CS-6508	Brasil	TRS-Color
Digitus	DGT-100	Brasil	TRS-80 Mod.III
Digitus	DGT-1000	Brasil	TRS-80 Mod.III
Digitus	DGT-AP	Brasil	Apple II +
Dismac	D-8000	Brasil	TRS-80 Mod. I
Dismac	D-8001/2	Brasil	TRS-80 Mod. I
Dismac	D-8100	Brasil	Apple II +
Dynacom	MX-1600	Brasil	TRS-Color
ENIAC	ENIAC II	Brasil	Apple II +
Engebras	AS-1000	Brasil	Sinclair ZX-81
Filcres	NEZ-8000	Brasil	Sinclair ZX-81
Franklin	Franklin	USA	Apple II +
Gradiente	Expert GPC1	Brasil	MSX
Houston	Houston AP	Brasil	Apple II +
Kemitron	Naja 800	Brasil	TRS-80 Mod.III
LNW	LNW-80	USA	TRS-80 Mod. I
LZ	Color 64	Brasil	TRS-Color
Magnex	DM II	Brasil	Apple II +
Maxitronica	MX-2001	Brasil	Apple II +
Maxitronica	MX-48	Brasil	Apple II +
Maxitronica	MX-64	Brasil	Apple II +
Maxitronica	Maxitronic I	Brasil	Apple II +
Microcraft	Craft II Plus	Brasil	Apple II +
Microcraft	Craft IIe	Brasil	Apple IIe
Microdigital	TK-3000 IIe	Brasil	Apple IIe
Microdigital	TK-82C	Brasil	Sinclair ZX-81
Microdigital	TK-83	Brasil	Sinclair ZX-81
Microdigital	TK-85	Brasil	Sinclair ZX-81
Microdigital	TK-90X	Brasil	Sinclair Spectrum
Microdigital	TKS-800	Brasil	TRS-Color
Milmar	Apple II Plus	Brasil	Apple II +
Milmar	Apple Master	Brasil	Apple II +
Milmar	Apple Senior	Brasil	Apple II +
Multix	MX-Compacto	Brasil	TRS-80 Mod.IV
Omega	MC-400	Brasil	Apple II +
Polymax	Maxxi	Brasil	Apple II +
Polymax	Poly Plus	Brasil	Apple II +
Prologica	CP-200	Brasil	Sinclair ZX-81
Prologica	CP-300	Brasil	TRS-80 Mod.III
Prologica	CP-400	Brasil	TRS-Color
Prologica	CP-500	Brasil	TRS-80 Mod.III
Ritas	Ringo R-470	Brasil	Sinclair ZX-81
Sharp	Hotbit HB-8000	Brasil	MSX
Spectrum	Microengenho I	Brasil	Apple II +
Spectrum	Microengenho II	Brasil	Apple IIe
Spectrum	Spectrum ed	Brasil	Apple II +
Suporte	Venus II	Brasil	Apple II +
Sycomig	SIC I	Brasil	Apple II +
Sysdata	Sysdata III	Brasil	TRS-80 Mod.III
Sysdata	Sysdata IV	Brasil	TRS-80 Mod.IV
Sysdata	Sysdata Jr.	Brasil	TRS-80 Mod.III
Timex	Timex 1000	USA	Sinclair ZX-81
Timex	Timex 1500	USA	Sinclair ZX-81
Timex	Timex 2000	USA	Sinclair Spectrum
Unitron	AP II	Brasil	Apple II +
Victor do Brasil	Elppa II Plus	Brasil	Apple II +
Victor do Brasil	Elppa Jr.	Brasil	Apple II +
Video Genie	Video Genie I	USA	TRS-80 Mod. I

INPUT foi especialmente projetado para microcomputadores compatíveis com as sete principais linhas existentes no mercado.
Os blocos de textos e listagens de programas aplicados apenas a determinadas linhas de micros podem ser identificados por meio dos seguintes símbolos:


Sinclair ZX-81


TRS-80


TK-2000


MSX


Spectrum


TRS-Color


Apple II

Quando o emblema for seguido de uma faixa, então tanto o texto como os programas que se seguem passam a ser específicos para a linha indicada.

NO PRÓXIMO NÚMERO

PROGRAMAÇÃO BASIC

O que são conjuntos e como dimensioná-los. A declaração **DIM**. Como utilizar os dados de um conjunto.

APLICAÇÕES

Digite os números e deixe o computador convertê-los em histogramas coloridos e atraentes.

CÓDIGO DE MÁQUINA

Como escrever em linguagem Assembler. Como converter os códigos. O que são mnemônicos.

PROGRAMAÇÃO BASIC

Aprenda a usar sprites, base do funcionamento de muitos jogos de ação.